SA/TR-2/89

A003 : FINAL REPORT

COMPUTER ALGORITHMS AND ARCHITECTURES FOR THREE-DIMENSIONAL EDDY-CURRENT NONDESTRUCTIVE EVALUATION

Contract No. N 00019-86-C-0219

with

Sabbagh Associates, Inc.
4639 Morningside Drive
Bloomington, IN 47401

for

Naval Air Systems Command

20 January, 1989

Volume III

CHAPTERS VI–XI

$\int_A$

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT: Approved for Public Release; Distribution is unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S): SA/TR-2/89	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: SABBAGH ASSOCIATES, INC. / 6b OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION: Department of The Navy, Naval Air Systems Command
6c. ADDRESS (City, State, and ZIP Code): 4639 Morningside Drive, Bloomington, IN 47401	7b. ADDRESS (City, State, and ZIP Code): ATTN: AIR-931A, Washington, DC 20361-9310
8a. NAME OF FUNDING / SPONSORING ORGANIZATION: Naval Air Systems Command / 8b OFFICE SYMBOL (If applicable): N00019	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: N00019-86-C-0219
8c. ADDRESS (City, State, and ZIP Code): AIR-931A, Washington, DC 20361-9310	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. 65502N / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO
11 TITLE (Include Security Classification): COMPUTER ALGORITHMS AND ARCHITECTURES FOR THREE-DIMENSIONAL EDDY-CURRENT NONDESTRUCTIVE EVALUATION. (Volume III - Chapters VI-XI)	
12 PERSONAL AUTHOR(S): SABBAGH ASSOCIATES	
13a. TYPE OF REPORT: Final / 13b. TIME COVERED: FROM 12/2/86 TO 1/20/89	14 DATE OF REPORT (Year, Month, Day): 1/20/89 / 15 PAGE COUNT: 180
16 SUPPLEMENTARY NOTATION	

17 COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Eddy-current nondestructive evaluation, Three-dimensional image reconstruction, inverse problems, conjugate gradient algorithm, computer architecture and DSP integrated circuits.
14	02		
11	04		

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

SEE NEXT PAGE

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL: Dr. L. E. Sloter	22b TELEPHONE (Include Area Code): (202) 692-7445 / 22c OFFICE SYMBOL: AIR-931A

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

18. Digital signal processing, image processing, classification

19. ABSTRACT

In this report, we develop an electromagnetic model for three-dimensional inversion of eddy-current data, an inversion algorithm based on the conjugate gradient technique, and a special purpose computer that we estimate can execute this algorithm in times comparable to high speed main-frames. This computer has a pipeline architecture and is designed around our parallel implementation of the inversion algorithm and makes use of high-speed DSP chips. The inversion process achieves a higher performance measure when more than one data set is inverted. The sequential order of the inversion scheme restricts the number of active elements in the pipe for a single problem. When more than one inversion problem enters the pipe, then more than one element could be active to improve the overall performance of the system.

The basic electromagnetic model starts with the integral equations for electromagnetic scattering, which are then discretized by means of the method of moments. This gives us the fundamental inversion model, which is then solved using the conjugate gradient algorithm. In order to accomplish the three-dimensional inversion, we acquire data at a number of frequencies; therefore, our inversion process is called a multifrequency method. The choice of frequencies, and the number of frequencies to be used, depend upon the conductivity of the host material, and the depth resolution sought.

The method of conjugate gradients has a number of attractive features for our purposes. Chief among them is that it allows a large problem to be solved efficiently, and, because it is an iterative algorithm, it allows us to take advantage of the special Toeplitz structure of the discretized model. We also derive an algorithm that allows us to constrain the solution, use preconditioning and a Levenberg-Marquardt parameter. Preconditioning is often useful in improving the convergence of the conjugate gradient algorithm, and the Levenberg-Marquardt parameter is needed to stabilize the solution against the effects of noise and modeling inaccuracies.

The inversion algorithms may require *a priori* information about the flaw regions. The information can be used to concentrate the inversion efforts on regions of interest rather than unflawed regions. Statistical pattern recognition and computer vision techniques have been examined to achieve this goal. The purpose of applying statistical pattern recognition techniques, is to detect the flaw regions and the background regions in the spatial domain. In addition, a graphical tool can be used to analyze the raw data when used as input features, and evaluate the classifiability of the measurement (any two features).

CHAPTER VI

CLASSIFICATION OF EDDY CURRENT MEASUREMENTS FOR FLAW DETECTION: LINEAR CLASSIFIERS

Bishara Shamee

1. Overview and Introduction

The inversion algorithms may require apriori information about the flaw regions. The information can be used to concentrate the inversion efforts on regions of interest rather than unflawed regions. Statistical pattern recognition and computer vision techniques have been examined towards this goal. In this chapter, we shall present the pattern recognition approach, while the second approach is presented in chapter IX.

Our goal of applying statistical pattern recognition techniques, is to detect the flaw regions and the background regions in the spatial domain. The regions identified as flaws, are the projection of the three-dimensional flaw unto the measurement plane. These planar regions correspond to the three-dimensional flaw only in the measurement plane and do not indicate the depth of the flaw. The three-dimensional conductivity profile is obtained by inversion of the measurements.

The decision will be based on a *feature* for each sample point. If the features show sufficient separability then the detection scheme would have low detection errors. Two class detection will be denoted by *binary detection* to indicate that only two classes are considered. The features used in this chapter are listed in chapter IX. Figure 1 below shows the block diagram of the classification process when the classifier parameters have been estimated. The estimate is derived from a training set extracted by the segmentation as discussed in chapter IX.

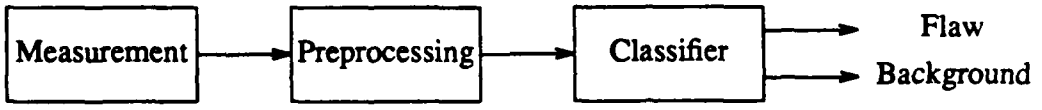


Figure 1: Block diagram of the flaw detection system for eddy current inversion.

We applied a multi-class classifier to the eddy current data when the number of classes is two. The multi-class can then be used (we have not pursued this idea) to detect and classify flaws among flaw classes. We believe that the feature set extracted from a flaw region, would contain sufficient information to identify flaws from one another. Such a system, in principle at least, would isolate the flaw regions into separate regions where each region would be associated with one type of flaw. The multi-class scheme was also motivated by the need to isolate the *tows*[1] present in the eddy current scan of some material. The tows were introduced as a third class in addition to flaws and background.

A graphical tool developed in this chapter to analyze the raw data when used as input features, and evaluate the classifiability of the measurement (any two features). The two-dimensional scatter diagram of any two features could also be obtained. Further more, each cluster of the data is represented by an ellipse reflecting the cluster relative size and orientation. Graph theory is used to obtain the class boundaries in the two-dimensional case by eliminating segments that do not belong to the classification boundary.

1 Appendix E, page 20 of the FIFTH QUARTERLY REPORT for this contract.

2. Multi-Class Piece-wise Linear Classifiers

With the introduction of tows in some measurements, a more elaborate scheme is required to detect the tows as well. In the identification problem (identifying regions corresponding to various flaws), the number of classes depends on the number of flaw types considered. For example, if the classifier is to identify two different kinds of flaws, then the number of classes is three: One for the background and two for the flaws.

When the multi-class problem is considered, the linear classification problem reduces to finding intersections of hyperplanes in the feature space. The dimension of the feature space will be reduced to two for part of this report to illustrate and test the algorithms developed. In later sections this constraint will be dropped and the full feature set in current use will be utilized. Further more, when only two features are used (real and imaginary part), the complexity of the classifier is decreased.

A linear classifier for two classes has the form:

$$h(x) = V^T x + V_o, \tag{1}$$

where V is the weight vector, V_o is the threshold, x is the feature vector and h is the decision function. The parameters of the linear classifier are determined during training by minimizing the error-probability. The sixth quarterly report showed some results for the two class case using a linear classifier. By increasing the number of classes, the classification or detection scheme can reveal more about the characteristics of the measured data. As the number of classes increases, the actual decision boundary is determined by the intersection of pair-wise hyperplanes in the feature space. The design will be followed closely to the piece-wise linear classifiers presented in [F1], and is derived by considering two classes at time.

Assume there are M classes [F1], then there are $(M)(M-1)/2$ discriminant functions, corresponding to each class pair. To each pair ij there is a linear separating boundary of the form:

$$h_{ij}(x) = V_{ij}^T x + V_{ijo} \qquad i \neq j \in \{1, 2, \ldots, M\}. \tag{2}$$

The signs of V_{ij} are selected such that the $i-th$ class distribution is on the positive side of h_{ij} and the $j-th$ class distribution lies on the negative side. A sample (or a measurement) x belongs to the $i-th$ class if $\{ h_{ij}(x) > 0: i, j = 1, 2, 3, \ldots M; i \neq j\}$. The $i-th$ class identification with the positive region in the feature space holds for connected regions and each class is represented by one cluster thus eliminating the situation where different clusters are of the same class.

The number of classes desired for classification is determined primarily by the number of distinct regions of interest. The decision function between the $ij-th$ pair is denoted by h_{ij} with the implicit understanding the $i \neq j$. The weight or coefficient vector is denoted by V_{ij} and the thresohld is V_{ijo}. The statistical characteristics of the data are obtained from the conditional means η_i, η_j, and covariance matrices Σ_i and Σ_j. The *a priori* probabilities are given by $P(\omega_i)$ and $P(\omega_j)$ and an estimate of these probabilities can be obtained from the segmentation results. The error probability in terms of the error function is denoted by ε_{ij}, while the actual miss-

classification count during training is denoted by e_{ij}. The expression for the error probability was derived in the sixth report for two classes and is stated in the multi-class formulation. The following is the complete formulation of multi-class linear classifier followed by an explanation of the steps involved:

Multi-Class Piece-Wise Linear Classifiers

For each $i = 1, 2, 3, 4, \ldots, M$

$$\sigma_i^2 = V^T \Sigma_i V,$$

For each $j = i + 1, i + 2, \ldots, M$

For $s \leftarrow s + \delta s; (0 \le s \le 1)$

$$V_{ij} = (s\Sigma_i + (1-s)\Sigma_j)^{-1} (M_j - M_i),$$

$$\sigma_j^2 = V^T \Sigma_j V,$$

$$V_{ijo} = -\frac{s\sigma_i^2 V_{ij}^T M_j + (1-s)\sigma_j^2 V^T M_i}{s\sigma_i^2 + (1-s)\sigma_j^2},$$

$$h_{ij}(x) = V_{ij}^T x + V_{ijo},$$

$$e_{ij}(x) = \begin{cases} 1 & \text{incorrectly classified} \\ 0 & \text{correctly classified} \end{cases}$$

$$e_{ij} = \sum_x e_{ij}(x)$$

$$\eta_i = V^T M_i + V_{ijo},$$

$$\eta_j = V^T M_j + V_{ijo},$$

$$\epsilon_{ij} = P(\omega_j) + \frac{1}{2} P(\omega_i)\, erfc\left(-\frac{\eta_i}{\sigma_i\sqrt{2}}\right) - \frac{1}{2} P(\omega_j)\, erfc\left(-\frac{\eta_j}{\sigma_j\sqrt{2}}\right)$$

$$\epsilon = \frac{1}{M} \sum_i \sum_j \epsilon_{ij}$$

The procedure outlined above yields the decision boundary between each pair with minimum error probability. The minimum error probability criterion used in this report is based on the error function and was derived for the two class case in the previous report. The actual miss-classification normalized count, is more accurate for a particular training set but may not yield the optimum value when considering samples for classification, since this fine tuning step may bias the classifier heavily towards a given training set.

Starting with the initial step, the variance of the i–th class is obtained as i varies from the first to the last class. The next step is to determine the resolution of the classifier parameter s. In this report, the resolution was set to $\delta s = 1/128$ meaning that the algorithm will find the linear classifier that gives the smallest error among 128 classifiers for each class pair. Note the weight vector V depends on the mean vector separation and the algorithm should be modified should

any two classes have equal or very close means. Two classes of one of the simulated data set had close mean vector, and as will be illustrated the error was higher than other class pair. The decision hyperplane between the two pairs is obtained and then the data is classified by knowing *a priori* the class to which the sample belong. This training would give us an actual error graph that could be compared to the theoretical error when the classes are assumed to be Gaussian. In a later section, results of the classification and error plots will be shown.

In this chapter, two synthesized data sets and actual data will be considered for evaluation and testing. Each class pair yield a decision boundary, and for a given class, the actual decision boundary is formed of combining other pair-wise hyperplanes. The test data set include a set with minimal class overlap, a set with similar properties for small separability, and actual data taken in the laboratory. The statistical properties of the sets used are shown in Table 1. The following section discusses the graphical representation when the feature space is restricted to two. It should be mentioned that this restriction has to be evaluated for real data followed by mapping higher dimensional feature space unto the two-dimensional feature space.

2.1. Two-Dimensional Graphical Representation

The graphical representation of the classes will be examined here to test the software and provide some examples for the detection process. The two-dimensional restriction will provide an evaluation of flaw detection with only two features specifically the measurement data.

There are a number of pair-wise linear segments forming the boundary region for a particular class resulting from the multi-class algorithm presented in section 2.0. For example, Figure 2 shows the cluster diagram of a two-dimensional, four class problem. The mean vectors and covariance matrices have been chosen to separate nicely for illustrative purposes with statistical properties shown in Table 1. The decision boundary is actually derived from the estimates of the statistical properties for the simulated and actual data.

Table 1: The characteristics of simulated data for program development

Statistical Properties of Simulated Data												
Data set *anti*												
Class 1			Class 2			Class 3			Class 4			
M_1	Σ_1		M_2	Σ_2		M_3	Σ_3		M_4	Σ_4		
4.0	0.7	0.0	0.0	0.7	0.0	-4.0	0.4	0.0	0.0	0.4	0.0	
0.0	0.0	0.4	4.0	0.0	0.4	0.0	0.0	0.7	-4.0	0.0	0.7	
Data set *two*												
Class 1			Class 2			Class 3			Class 4			
M_1	Σ_1		M_2	Σ_2		M_3	Σ_3		M_4	Σ_4		
7.82	1.03	1.28	5.76	4.79	4.41	6.61	1.63	2.15	6.12	5.11	4.73	
6.75	1.28	1.96	5.71	4.41	5.07	5.06	2.15	3.59	6.28	4.73	5.68	

Each class in the cluster diagram of Figure 2 will be replaced by an ellipse characterizing the class as illustrated in Figure 3 on the following pages. The parameters of the ellipse are determined from the eigenvalues and eigenvectors of the whitening transformation discussed in the previous report as well as in the next section. The boundary of any ellipse in Figure 3, show the orientation and the relative variance size of a Gaussian distribution with identical mean and covariance matrix as the estimated. Some of the samples would lie outside the ellipse boundary, hence the boundary is a probabilistic characterization of the particular class. In later sections, the probability that a given sample may lie within the boundary will be derived.

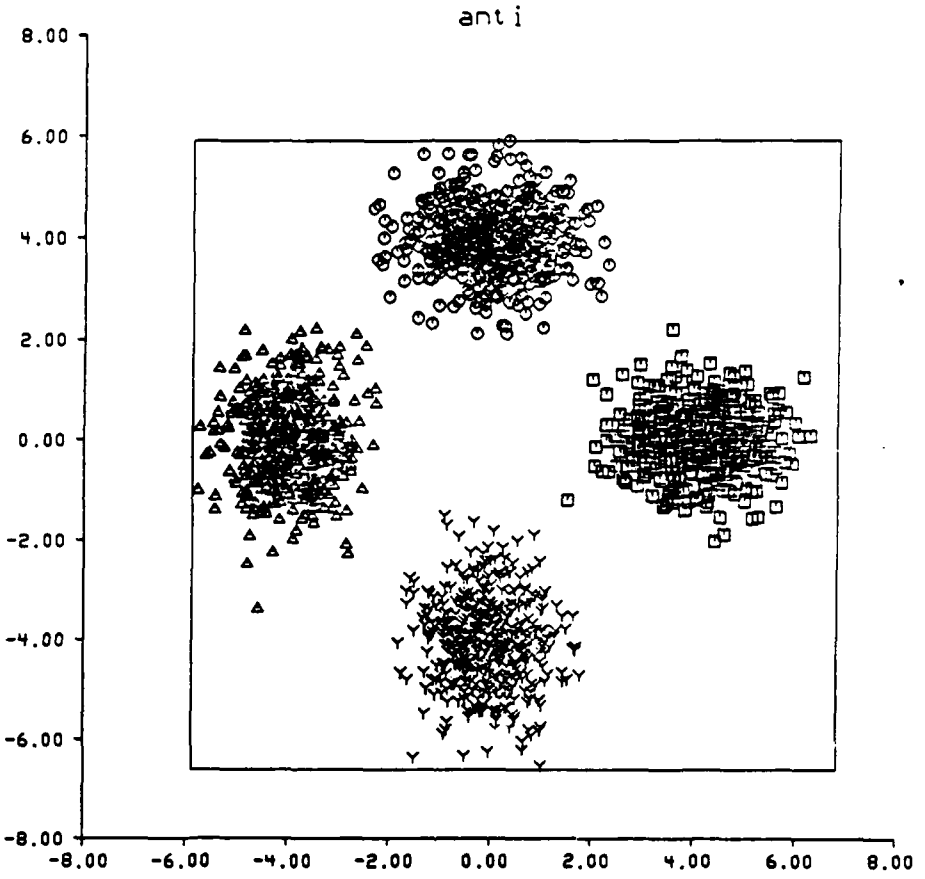


Figure 2: Scatter diagram of the simulated data for the parameters listed in Table 1.

The straight lines shown in Figure 4 are the corresponding pair-wise linear classifiers obtained by considering two classes at a time. Each straight line is labeled by the actual pair it divides. For example: h_{12} is the classifier that divides classes one and two, h_{23} is the classifier that divides classes two and three, and h_{ij} is the classifier that divides the i-th and j-th class. A feature vector x is assigned to the $i-th$ class if:

$$h_{ij}(x) > 0 \quad j \neq i. \tag{3}$$

Note that not all the line segments shown in Figure 4 belong to the decision boundary, because the decision boundary is composed of the intersection of pair-wise regions. In order to track and represent the boundary graphically, the following algorithm is presented. It follows the idea of boundary tracking in digital images as discussed in earlier reports under component labeling.

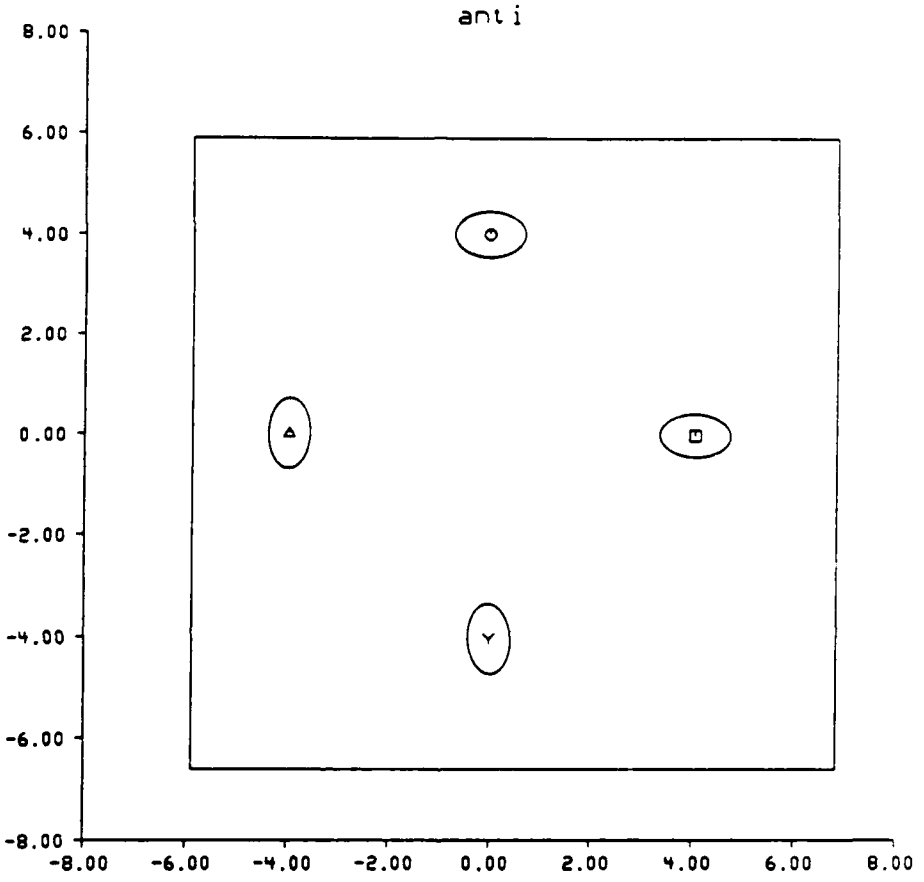


Figure 3: The characteristic ellipses of the simulated clusters shown in Figure 2. The center of the ellipse correspond to the estimated mean of data set. The major and minor axis denote the principal orientation of each cluster.

2.2. Graphical Representation of Classes

In this section, a scheme to represent each class by an ellipse is presented. The elliptical representation is quiet arbitrary and is preffered because the principal components of the classes can be alligned with the major and minor axis of the ellipse. Furthermore, the equal probability curves of a Gaussian density are ellipses whose parameters are closely related to the covariance matrix. For the Gaussian case, it is sufficient to know the mean vector and the covariance matrix to determine the ellipse that represents the distribution.

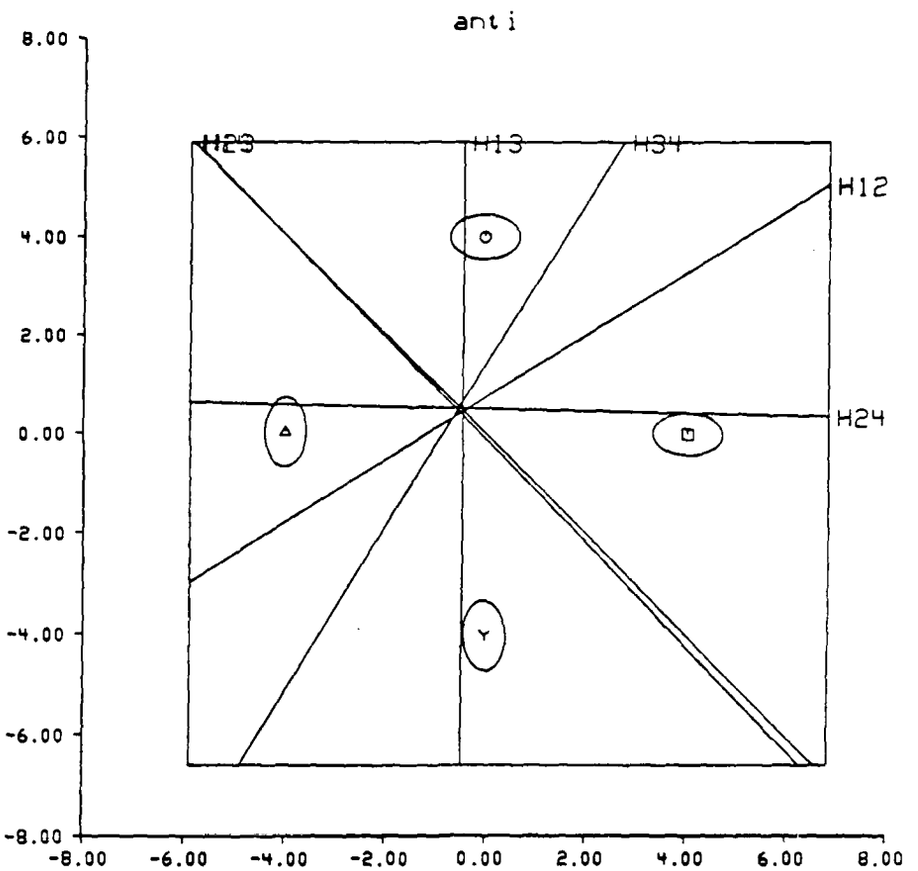


Figure 4: The decision boundaries for each class pair of the simulated data. Each line is labeled by the class pair it separates. Not all the line segments belong to the decision boundary.

Since the ellipse forms a region in the feature space, integrating the density function over this region yields the probability that a given sample falls in the region. Thus, for a given probability, an ellipse can be chosen to represent the classes. However, the density function is not known a priori which will make it difficult to obtain the probability since an estimate of the density may be required. In this case (unknown density), an estimate of the probability can be obtained by counting the number of samples inside the ellipse (or region) and dividing by the total number of samples in the class. As the parameters of the ellipse vary, an estimate of the probability as a function of the ellipse parameters is obtained. The ellipse is chosen based on a given probability level. For example, in the familiar one-dimensional Gaussian case, if a level of 68% is chosen, then the interval (width of the ellipse) would be one standard deviation on each side of the mean, and for a 95% level, the width of the interval would be two standard deviation on each side of the mean.

The initial step in this representation is to find the principal components of the class under consideration. The *center* of the class is the estimated mean[1], thus by shifting the coordinate

1 The estimated mean is used in all the classification experiments performed in this report. This includes the simulated and the actual data.

system in the feature space, the class would have zero mean. Next, the principal components of the distribution are determined. These components are determined via an *Orthonormal* transformation as done below for the Gaussian case [F1].

Suppose the feature vector is drawn from a jointly Gaussian distribution with density (after subtracting the mean) given by:

$$f(X) = (2\pi)^{-n/2} \, |\Sigma|^{-1/2} \exp(-\frac{1}{2} X^T \Sigma^{-1} X), \tag{4}$$

where X is a two-dimensional feature vector and Σ is the class covariance matrix. The term defined for the exponential in the density of equation (4) is a distance metric. It measures the distance from the mean (origin here) to any point X in the feature space weighted by the covariance matrix. The principal components of the distribution are determined by finding the vector X which maximize (or minimize) the weighted distance metric with constant magnitude. That is, find the vector X which maximizes $X^T \Sigma^{-1} X$ subject to $X^T X = c$, where c is a constant. Proceeding with defining the Lagrangian multiplier μ, the cost function becomes:

$$f(X, \mu) = X^T \Sigma^{-1} X - \mu (X^T X - c), \tag{5}$$

and upon taking derivatives the following necessary conditions are obtained:

$$\Sigma^{-1} X - \mu X = 0. \tag{6}$$

By defining $\lambda \equiv 1/\mu$, the necessary condition is equivalent to $\Sigma X = \lambda X$ which for non-singular covariance matrix reduces to solving the characteristic equation:

$$|\Sigma - \lambda I| = 0. \tag{7}$$

The covariance matrix Σ is real, symmetric and non-singular which implies that its eigenvalues are real. To each eigenvalue $\lambda_1, \lambda_2, \ldots, \lambda_n$, an eigenvector $\phi_1, \phi_2, \ldots, \phi_n$ is determined by solving $\Sigma \phi_i = \lambda_i \phi_i$. The set of eigenvectors are orthogonal which is shown below.

Let λ_i and λ_j be any two eigenvalues such that $i \neq j$ with the corresponding eigenvectors ϕ_i and ϕ_j satisfying the following conditions:

$$\Sigma \phi_i = \lambda_i \phi_i, \tag{8}$$

$$\Sigma \phi_j = \lambda_j \phi_j. \tag{9}$$

By multiplying equation (8) by ϕ_j, equation (9) by ϕ_i and subtracting the resulting equations to give the following:

$$(\lambda_i - \lambda_j)\, \phi_i^T\, \phi_j = \phi_j^T\, \Sigma\, \phi_i - \phi_i^T\, \Sigma\, \phi_j. \tag{10}$$

The right hand side of equation (10) is zero since Σ is a symmetric matrix. For distinct eigenvalues ($\lambda_i \neq \lambda_j$), the left hand side becomes $\phi_i^T\, \phi_j = 0$. Hence the eigenvectors are orthogonal and form the principal components of the class distribution.

The norm of each eigenvector is given by $\phi_i^T\, \phi_i$. Using the constraint equation, $X^T\, X$, when X is an eigenvector gives a norm of the constant c. By choosing the constant c to be a unity the eigenvectors form and orthonormal set. That is $\phi_i^T\, \phi_j = \delta_{ij}$, where δ_{ij} is the *Kronecker* delta function. Since the eigenvectors now define an orthonormal set, a new matrix Φ whose columns are the eigenvectors can define a linear transformation. That is,

$$\Phi = [\, \phi_1, \phi_2, \ldots, \phi_n \,]. \tag{11}$$

Equation (10) can then be written in terms of Φ with

$$\Sigma\, \Phi = \Phi\, \Lambda, \tag{12}$$

where Λ is an n x n diagonal matrix given below:

$$\Lambda = \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{bmatrix}. \tag{13}$$

The ortho-normality relation $\phi_i^T\, \phi_j = \delta_{ij}$ can be expressed in terms of the eigenvector matrix Φ as shown below:

$$\Phi^T\, \Phi = I, \tag{14}$$

where I is the identity matrix. Note that this equation indicates that the inverse of Φ is its transpose (i.e. $\Phi^{-1} = \Phi^T$). Thus equation (12) above can be written as:

$$\Phi^T\, \Sigma\, \Phi = \Lambda. \tag{15}$$

The eigenvector matrix, Φ, defines a linear, non-singular, ortho-normal transformation. This transformation would map the original distribution into the center and could be used to *whiten* the distribution. It will be used here to derive a characterization of the cluster relative to

each other as done next.

Let $Y = \Phi^T X$ be a new random vector derived from the original distribution X by applying the linear transformation just found. The mean of Y is zero as shown below:

$$M_y = E[Y] = E[\Phi^T X] = \Phi^T E[X] = 0, \tag{16}$$

since the distribution of X has been shifted by its mean. The covariance matrix Σ_y is:

$$\Sigma_y = E[(Y - M_y)(Y - M_Y)^T] = E[\Phi^T X X^T \Phi] = \Phi^T \Sigma \Phi. \tag{17}$$

By using equation (15) the covariance matrix of Y is the eigenvalue matrix, i.e.,

$$\Sigma_y = \Phi^T \Sigma \Phi = \Lambda. \tag{18}$$

The linear transformation defined by Φ transforms the original Gaussian distribution into a Gaussian distribution with a diagonal matrix whose elements are the eigenvalues of the original distribution. Since the covariance matrix of Y is diagonal, then the components of Y are uncorrelated random variables (for Gaussian case, they are also independent). Further more, the transformation could be used to diagonalize a given distribution and for our purposes in the graphical representation of the classes it allows to locate the pirincipal axis of a given class.

The eigenvectors of the linear transformation just derived, are orthogonal and could be used as a new coordinate system. In this coordinate system, the distribution of a Gaussian distribution is also Gaussian and is given by:

$$f(Y) = (2\pi)^{-n/2} |\Lambda|^{-1/2} \exp\{-\frac{1}{2} Y^T \Lambda^{-1} Y\}, \tag{19}$$

and for a given region in the feature space, the probability that a given sample falls in the region is just the integral of the probability density function over the region. In the following paragraphs, the dimension of the feature space is set to two in order to find explicit formulas for the graphical representation of the classes.

The ellipse parameters (major, minor axis and orientation) are chosen from the diagonal eigenvalue value matrix. The ellipse representing the classes will be called the *characteristic ellipse* and the major and minor axis are the variances in each eigenvector direction as illustrated in Figure 5 below.

The orientation of the ellipse are obtained from the principal directions ϕ_1 and ϕ_2. It is also desired to evaluate the probability that a given sample would fall in the characteristic ellipse. If y_1 and y_2 are the coordinates in the transformed domain, the density of the class is then:

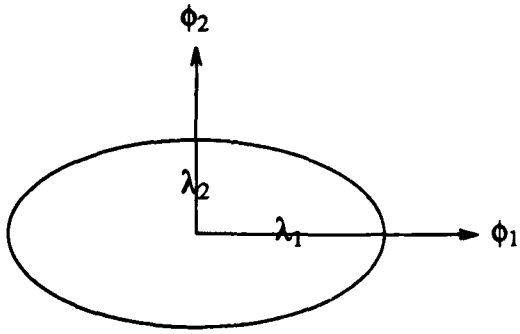


Figure 5: Characteristic ellipse: The axis determine the size and the orientation of the distribution.

$$f(y_1,y_2) = \frac{1}{2\pi\sqrt{\lambda_1\,\lambda_2}} \exp\{-\frac{1}{2}\,(\frac{y_1^2}{\lambda_1} + \frac{y_2^2}{\lambda_2})\}, \tag{20}$$

with the characteristic ellipse defined by:

$$\frac{y_1^2}{\lambda_1^2} + \frac{y_2^2}{\lambda_2^2} \leq 1. \tag{21}$$

The probability that a given class sample falls in the characteristic ellipse is then:

$$P(Y \in R) = \int_{-\lambda_1}^{\lambda_1} \int_{-\lambda_1\,(1-y_2^2/\lambda_2^2)^{1/2}}^{\lambda_1\,(1-y_2^2/\lambda_2^2)^{1/2}} f(y_1, y_2)\, dy_2\, dy_1, \tag{22}$$

where R is the region defined by Equation (21) and $Y = (y_1, y_2)$. The probability P is then a function of the eigenvalues, i.e., for given eigenvalues, the integral can be evaluated numerically, or could be approximated by counting the number of samples inside the characteristic ellipse divided by the total number of class samples. Figure 7 shows the cumulative probability over the elliptical regions as a function of the ellipse major and minor parameter. The lines of constant probability are shown in Figure 8. These lines, show that for a sample to fall within 99% of the characteristic ellipse, the major and minor axis should be enlarged to be around 10 times as their current value which is one standard deviation. The probability that a sample falls in this region is about 48% for the current setting (one standard deviation).

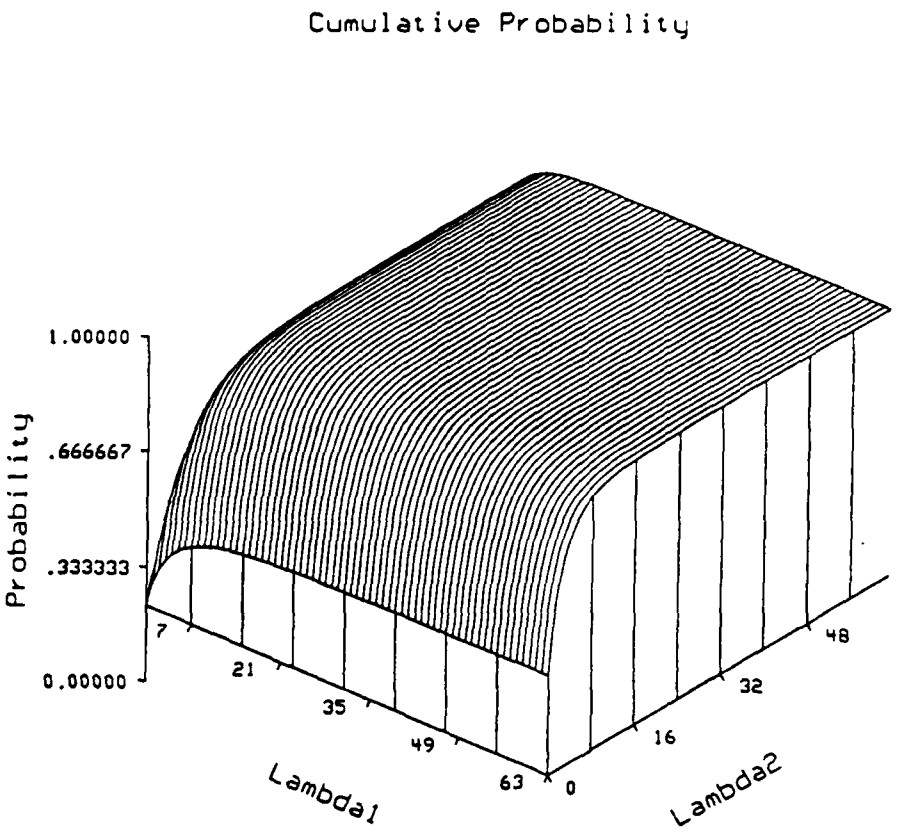


Figure 6: The Cumulative Probability function as a function of the eigenvalues.

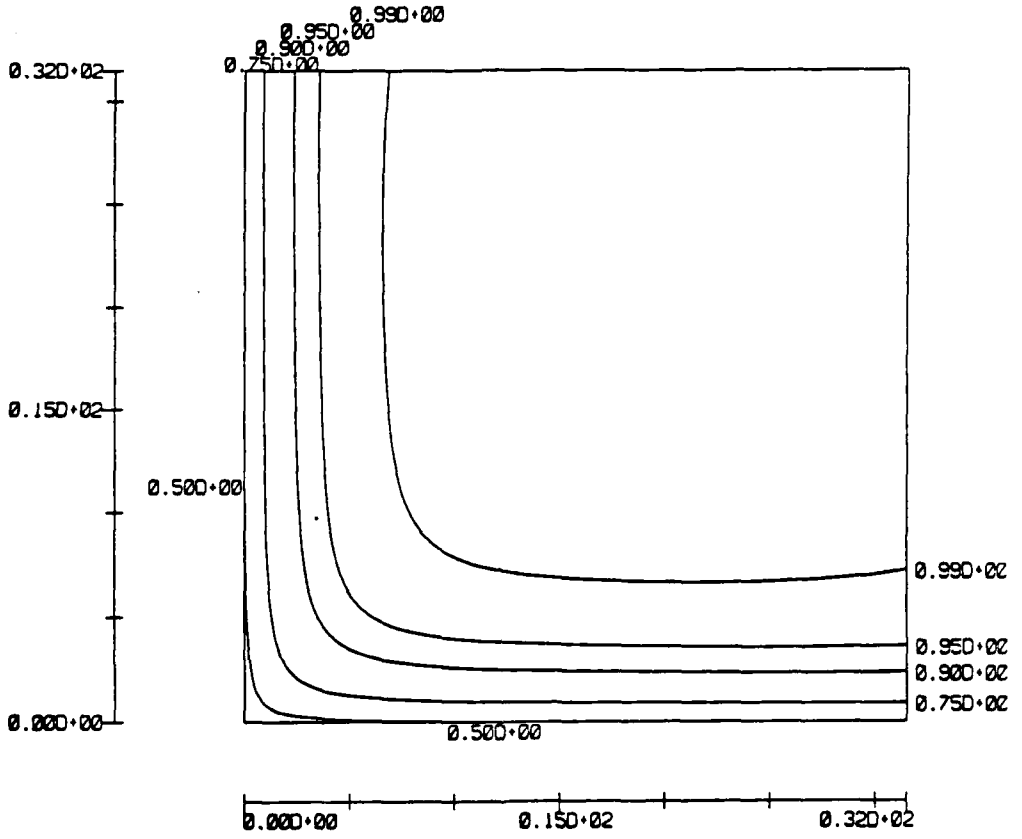


Figure 7: Level curves of the Cumulative Probability function.

2.3. Tracking Piece-Wise Linear Decision Boundaries

A generalized algorithm for tracking linear boundary will be devised here. The problem will be formulated independent of the previous section and applied to the classification problem.

Assume the planar region of interest is rectangular and bounded by $x_{min} \le x \le x_{max}$ and $y_{min} \le y \le y_{max}$ denoted by R. Each line $l(x,y)$, which separates two classes, intersects the region R in more than two points because the region R is bounded by the extremes of the data. This is not a severe assumption since the data is scanned *a priori* to determine the bounds of the region R. Every line now divides the region R into two regions corresponding to the sign of $l(x,y)$ denoted by R^+ and R^-. Let $R_i^+ = \{(x,y): l_i(x,y) > 0\}$ and $R_i^- = \{(x,y): l_i(x,y) < 0\}$. A *constraint set* is a list of inequalities defining a subset of the region R. The constraint list can describe the boundary of a given class. For example, Figure 9 shows a region R and three line

segments intersecting R. The three lines intersect the region in more than two points and may intersect each other. The intersection of these lines define small regions in terms of a constraint lists.

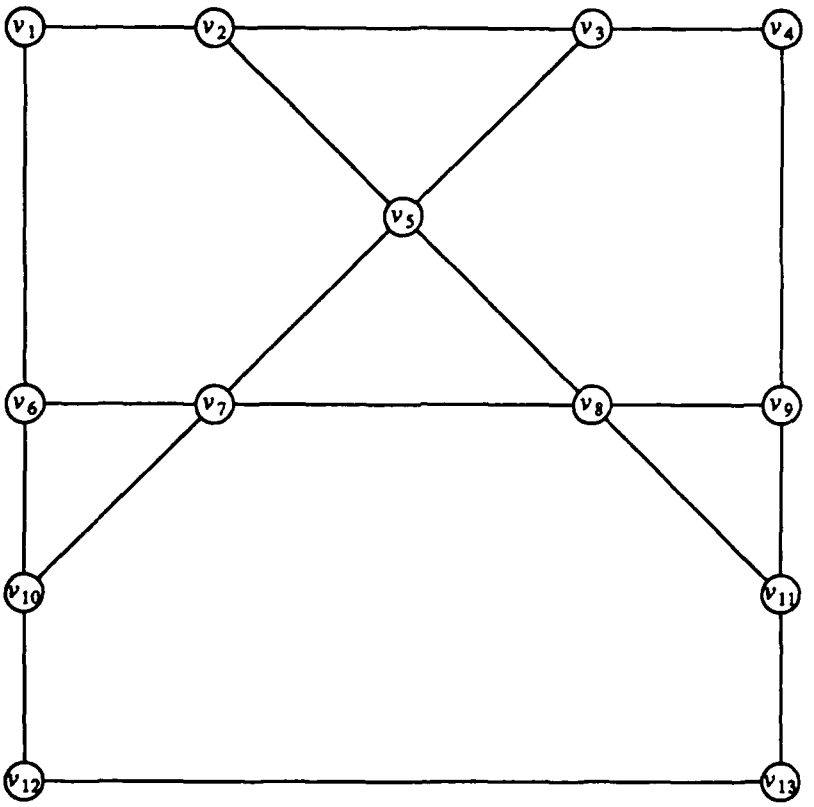


Figure 8: An example of vertex and region assigment for three class problem.

Each intersection point has been labeled including those of the boundary of the region R. For example, R_1^+ and R_2^- define the subset bounded by the line segments (v_8, v_9), (v_9, v_{11}) and (v_{11}, v_8). The boundary problem can be formulated as:

> Given straight lines $l_1, l_2, \ldots, l_n$ intersecting a rectangular, planar, and bounded region R. Each line intersects the region R in more than two points thus dividing R into regions. For a given constraint set, locate the line segments that define the boundary a class specified by a constraint list.

The problem of tracking the decision boundary can be formulated in terms of graph theory. Most of the definitions and results are well known in graph theory and are included here for completeness. An excellent reference is [B1] and for practical applications the reader is refered to [G1] and [A1]. Some definitions will be stated in order to formulate and solve the problem.

Definition: A *graph* is a set of points called *vertices* interconnected by a set of *edges*. The vertex set is denoted by V and the edge set by E and denote the graph by $G = (V,E)$. For example Figure 9 is a graph $G = (\{v_1,v_2,v_3,v_4\},\{e_1,e_2,e_3\})$.

A graph G is *finite* if the number of edges and the number of vertices is finite, and is *directed* if a direction is assigned to the edges. The edges in a graph can be specified in terms of the vertices they connect. The edges in Figure 8 can be written as: $E = \{(v_1, v_2),(v_1, v_4),(v_2, v_3)\}$

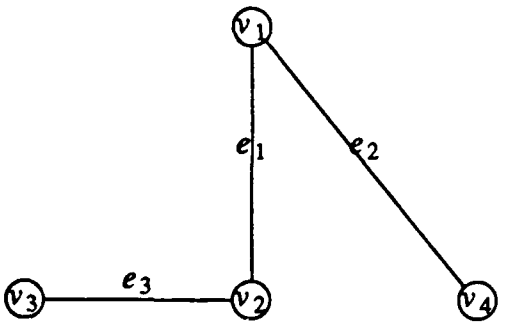


Figure 8: A graph with four vertices and three edges.

If an edge e has a vertex v as an endpoint, then e is *incident with* v, and if $(u,v) \in E$ then u is said to be *adjacent* to v. A *path* from v_1 to v_i is a sequence $P = v_1, e_1, v_2, e_2, \ldots, e_{i-1}, v_i$ of alternate vertices and edges such that for $1 \leq j < i$, e_i is incident with v_j and v_{j+1}. Two vertices v_i and v_j are *connected* if there is a path from v_i to v_j.

The basic and minimum definitions have been introduced, and any new concept will be defined as the material is covered. In order to solve problems using the computer, a suitable data structure is necessary to represent graphs. In this report, a suitable data structure will be implemented in order to track the linear boundary for graphical representation of multi-class linear classifiers. First, the most common data structures are presented.

An *Adjacency matrix* for a graph G is an n x n matrix such that:

$$A(i,j) = \begin{cases} 1 & (i,\ j) \in E \\ 0 & \text{otherwise} \end{cases}$$

That is if an edge is incident with vertices v_i and v_j then $A(i,\ j) = 1$, otherwise $A(i,\ j) = 0$. Thus the adjacency matrix A is symmetric for *undirected graphs* as illustrated for the graph of Figure 9 below:

While the matrix representation of a graph is conceptually easy, it requires large storage space and it is rather difficult to integrate in software. Another corresponding important data structure is the *adjacency list* of a graph. In an adjacency list, each vertex has an association list to its adjacent vertices. Two arrays are needed to represent the adjacency list, one for the labels, and another to keep track for the next adjacent vertex (The $\varnothing$ symbol denotes the end of the list). For example

A	v_1	v_2	v_3	v_4
v_1	0	1	0	1
v_2	1	0	1	0
v_3	0	1	0	0
v_4	1	0	0	0

Figure 9: The Adjacency matrix of example graph. If an edge (undirected in the example) exists between the $i-th$ and $j-th$ vertices, then $A(i,j) = A(j,i)$ is set to 1, else it is 0.

(see Figure 11), to obtain the set of all vertices adjacent to the vertex labeled 1, start with "1" and the next vertex is NEXT[1], followed by NEXT[NEXT[1]] and so forth. HEAD[NEXT[1]] will be the label of next vertex adjacent to "1". Some software languages (such as C) permit the use of a *pointer* to a structure which makes the adjacency list rather attractive without the need for array representation. Figure 11 shows the array representation for the adjacency list of the example graph.

	Head	Next
1	1	5
2	2	7
3	3	8
4	4	9
5	2	6
6	4	∅
7	3	∅
8	2	∅
9	1	∅

Figure 10: Array representation of the adjacency list for the graph of Figure 8, and the corresponding Adjacency matrix in Figure 9.

The complexity of the adjacency matrix graph representation is $O(n^2)$, while for the adjacency list is $O(n)$. Hence, based on complexity considerations, the adjacency list data structure for graph representation will be used in order to find the class regions for multi-class problems.

The first step is to construct the graph from the lines separating the pair-wise classes. A separating line will be identified by l_{ij} to indicate that this line separates the $i-th$ and $j-th$ classes. The same line will also be denoted by $l_{i \cdot M + j}$ depending on the fact that it separates two classes is relevant to the current discussion. The boundary of the region R is determined by scanning the data for the extremes defined by the set $R = \{(x, y): x_{min} \le x \le x_{max}, y_{min} \le y \le y_{max}\}$. The boundary of R will be included in the tracking algorithm, since no point lies outside R.

The vertices of the graph are determined by finding the intersection of all pairs of lines. Some of the intersection points lie outside the region R and will not be included in the algorithm. The following mathematical description will summarize the process of locating the internal vertices. Loosely, a vertex is internal if it falls inside or on the boundary of the region R.

Let $l_i(x, y)$ and $l_j(x, y)$ be any two lines including the four lines forming the boundary of the region R. The coordinates of the intersection point (if any) is located by solving a linear system of two equations. If the lines are given by:

$$\begin{aligned} l_i&: a_i x + b_i y + c_i = 0 \\ l_j&: a_j x + b_j y + c_j = 0. \end{aligned} \tag{23}$$

The point of intersection is obtained by solving the above system for x and y. The explicit solution is:

$$x = \frac{1}{\delta}(b_j c_i + b_i c_j), \quad \delta \neq 0, \tag{24}$$

$$y = \frac{1}{\delta}(a_i c_j - a_j c_i), \quad \delta \neq 0, \tag{25}$$

where $\delta = (a_i b_j - a_j b_i)$. If $\delta = 0$, the two lines are either identical or they do not intersect. It is straight forward to isolate the two cases for $\delta = 0$, each line is evaluated at a given x or y and compared to the other line.

Having determined the coordinates of the intersection point between a pair of lines, this intersection point will be declared a *vertex* if the coordinates are interior to the region R. The corner points of the region R, are declared vertices since the boundary of R is included in the algorithm.

The adjacency list is constructed by checking if their is an edge between any pair of vertices. An edge will exist if both vertices lie on some line. This is accomplished by finding the lines each vertex lies on. Each vertex is an intersection point and thus the lines defining the vertex are associated with the vertex. The lines associated with each vertex are compared to determine the existence of an edge between these two vertices.

After constructing the adjacency list for the graph, each vertex is tested for class membership. A vertex v belongs to the $i-th$ class if the following holds:

$$l_{ij}(v) > 0, \quad j = 1, 2, \ldots, M; j \neq i. \tag{26}$$

All the vertices satisfying the criterion in Equation (26) belong to the $i-th$ class and are collected as a list for the $i-th$ class. This list may contain un-necessary vertices which should be eliminated as discussed below.

Three or more of the vertices for a given class may be collinear. The collinearity introduces redundant line segments. If three vertices are collinear, then the middle vertex is eliminated. Only

three vertices are tested for collinearity at a given time. The test for collinearity is obtained by evaluating a 3 x 3 determinant. Let $v_i = (x_i, y_i)$, $i = 1, 2, 3$ be the coordinates of the three vertices under testing. Now, if the three points are collinear, then there is a straight line L defined by $a\,x + b\,y + c = 0$ passing through the three vertices which gives:

$$\begin{aligned} a\,x_1 + b\,y_1 + c &= 0 \\ a\,x_2 + b\,y_2 + c &= 0 \\ a\,x_3 + b\,y_3 + c &= 0, \end{aligned} \tag{27}$$

a homogeneous system of linear equations. A solution (a, b, c) will exist provided that the following determinant:

$$\delta = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix}, \tag{28}$$

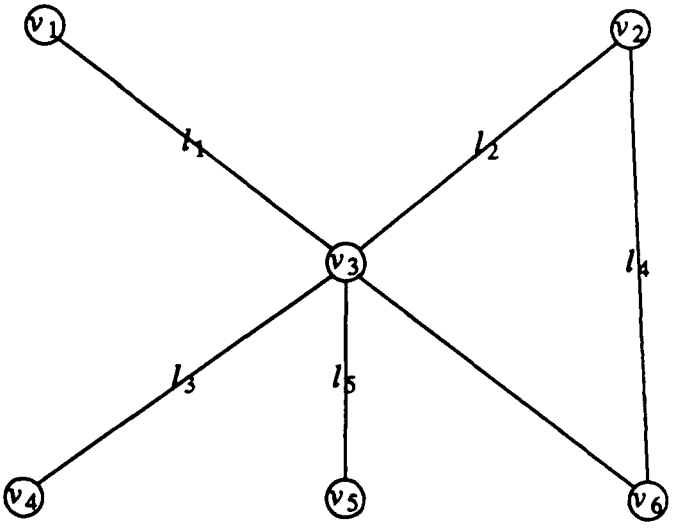


Figure 11: If three vertices of the same class are collinear, the middle vertex is deleted.

is zero and the three points are collinear. For example, as in Figure 11, if vertices v_1, v_3 and v_6 are vertices of class 1, and if they are collinear, then v_3 is deleted from the list of class1. The deletion of a vertex from a given class implies that all edges incident to that vertex are also deleted. Observe that the line segment between v_1 and v_6 still exists even though the vertex v_3 has been deleted because the edge v_6, v_1 was defined between vertex v_1 and v_6. If a vertex is deleted from a given class vertex list, it may appear if it is the vertex of any other class. Hence, a vertex is

eliminated from the graph if it is the middle vertex of some line segments in all the classes.

After deleting the middle vertices from a given class, the resulting vertices (together with the incident edges) form a graph which we shall denote by a *class graph*. The class graph may contain separate connected components and one of these components will define the boundary of the class. The search for a connected component is standard in graph theory and thus the algorithm will be presented for completeness. The reader may consult [H1] and [G1] for further information.

The connected component algorithm is a two part program, the first is a *depth first search* denoted by *dfs* and the second is called *connected* given below:

```
dfs(v)
{ Given a vertex v and a class graph. Dfs will return the sequence of adjacent ver-
tices. Initially the array visited is set to false.}
visited[v] = true;
for each vertex w adjacent to v do
        if w is not visited then dfs(w);
end for


connected(G)
{ Given a Graph G, with n vertices, connected finds the connected components of G
}
for i = 1, 2, . . ., n
      visited[i] ← false
for i = 1, 2, . . ., n
      if visited[i] == false then   dfs(i)
      end
end of connected()
```

The graph scanning algorithms, with the middle vertex deletion would form the core of the process of eliminating unwanted lines. Figure 12 is the result of applying these techniques to the graph found in Figure 4. Note that in Figure 12, there appears a small region that does not belong to any class. This region is usually called *rejection region* because it cannot be classified in any class. This region is usually small and cannot be eliminated when using linear classifiers. Of course one would declare the rejection region to be included in any class, but it will result in higher classification error in addition to the difficulty in assigning it to any class. Thus we will treat the rejection region as a region where no decision is possible.

The following few pages will cover the classification of four classes for the two simulated data sets given in Table 1. The same classification algorithms will be applied to actual data and similar plots will be obtained. For the case of actual data, images relating each cluster to the spatial location. After these figures Fisher classifiers is presented.

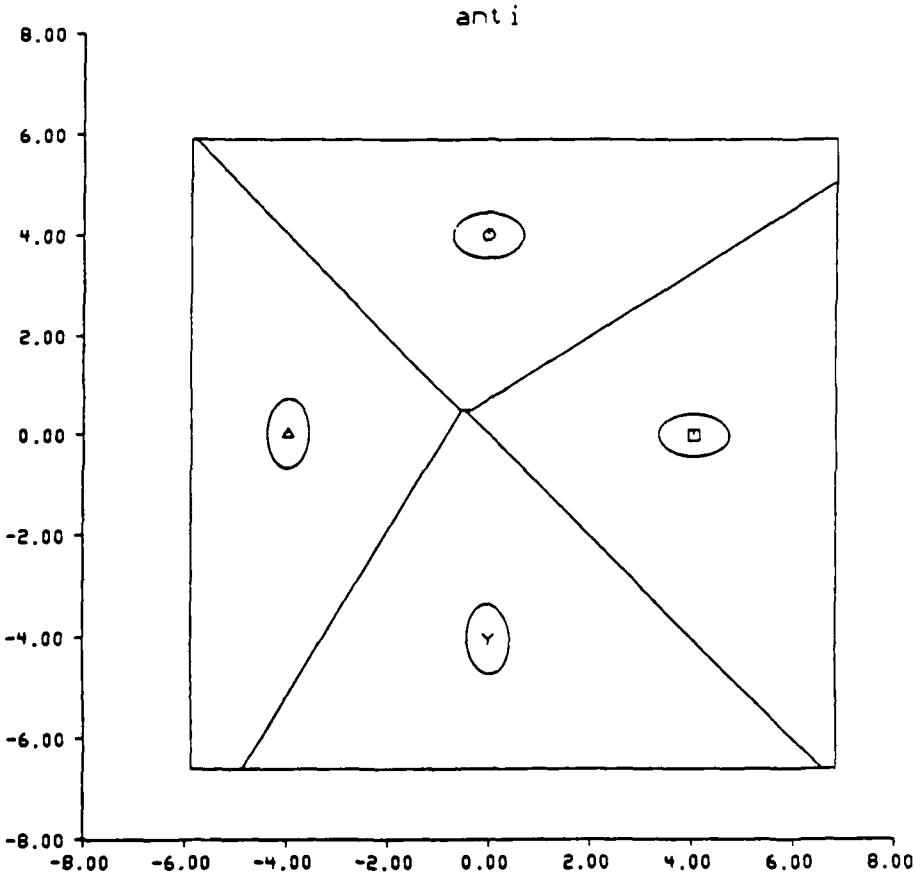


Figure 12: The result of the boundary tracking algorithm for obtaining the actual decision boundary of the simulated data. The small region around the center is the rejection region (i.e. the region that belongs to more than one class).

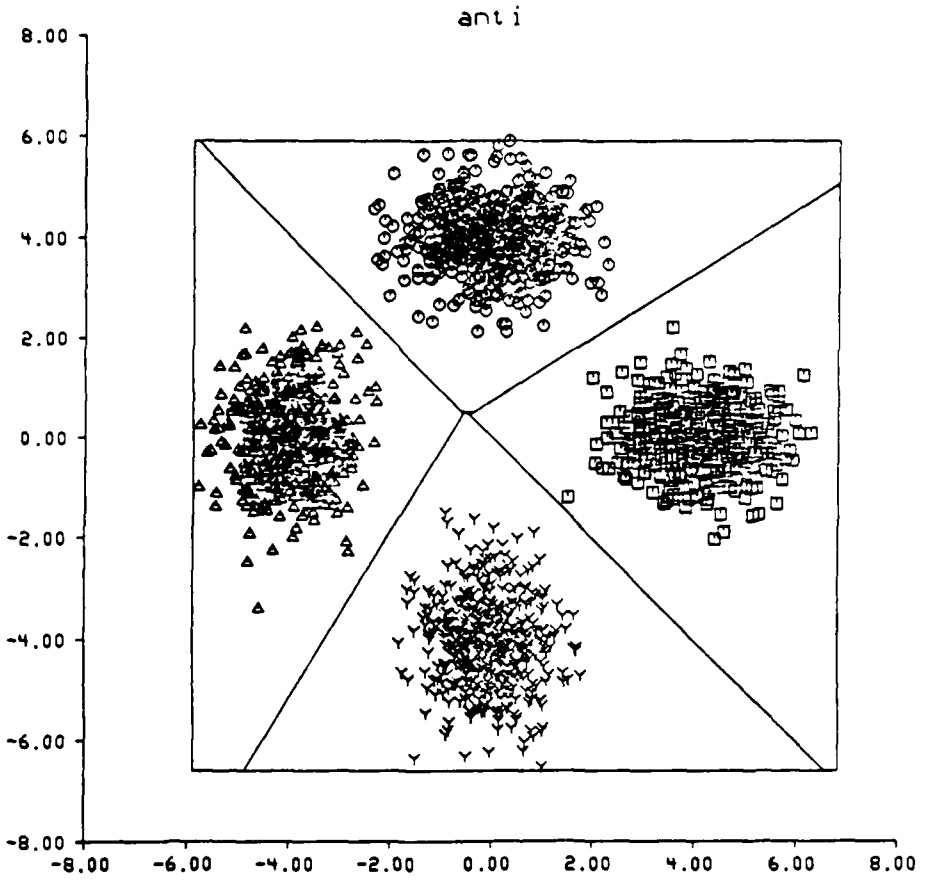


Figure 13: The decision boundary and the simulated data. In general, the actual data will not be well separated as illustrated here.

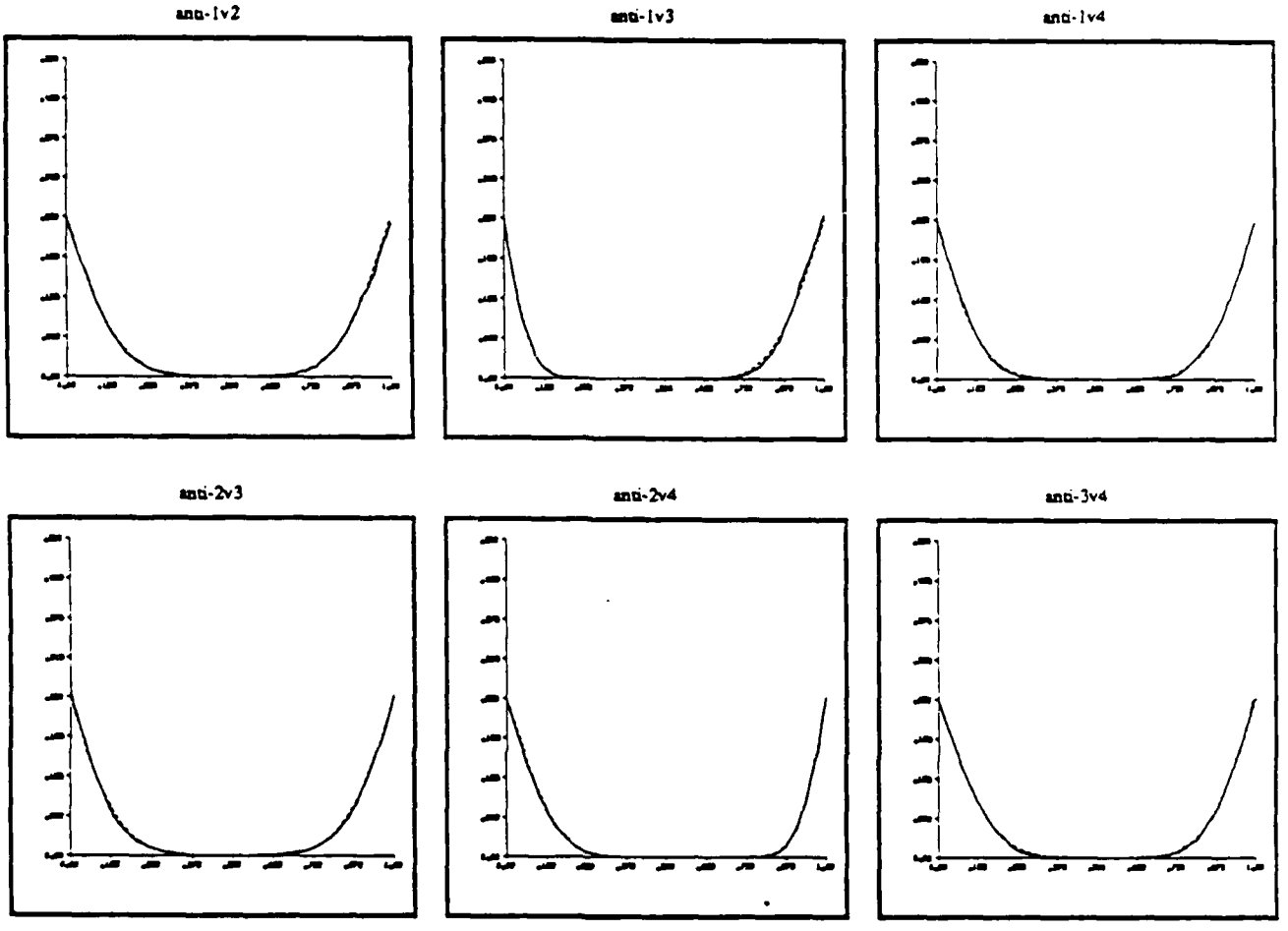


Figure 14: Pair wise classification error of the simulated data set as a function of classifier parameter *s*. The decision boundary in the previous two Figures correspond to the minimum pair wise error.

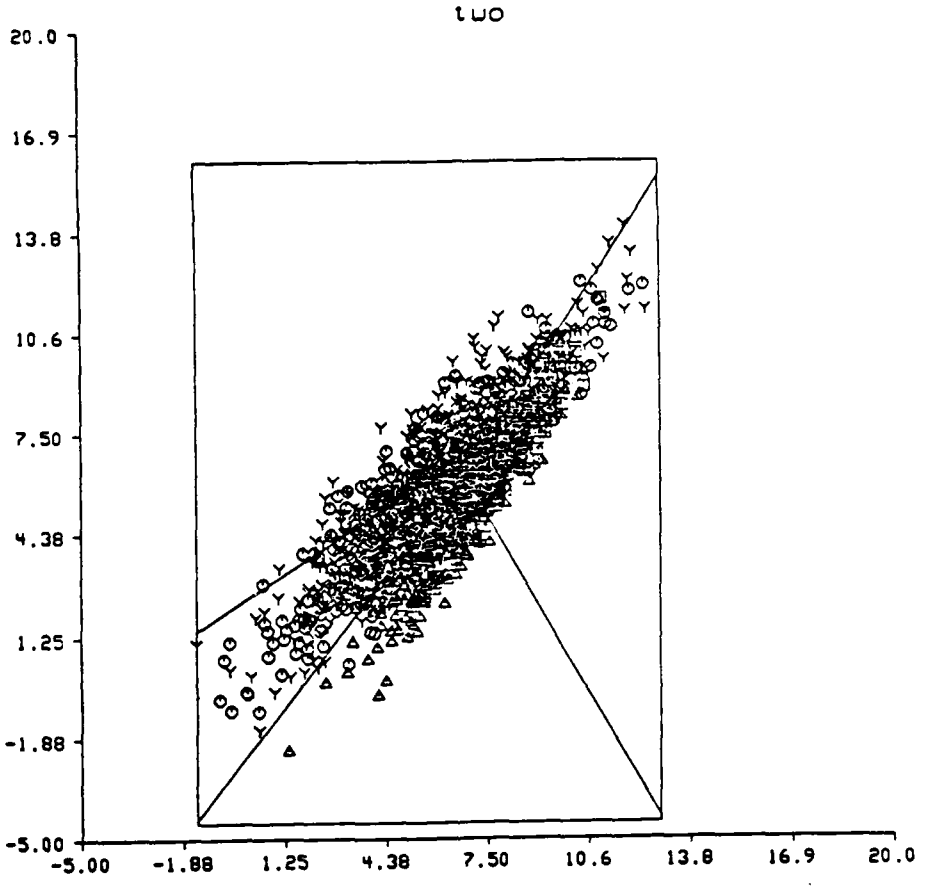


Figure 15: The *two* simulated data cluster diagram with the decision boundary.

3. Fisher Discriminant Criterion

A draw back to the multi-class linear classifier discussed in the previous section is its time complexity. For each class pair, as s varies over its range ($\delta s = 1/128$), 128 classifiers are designed and the miss-classification error is computed for each classifier. As the number of classes and feature dimension increases, the time increases drastically[1]. Another guide-line for the design is to find the weight vector V and the threshold V_o by minimizing an

[1] There are $128\,M!/(2!)\,(M-2)!$ classifiers being tested; The classifier with the minimum miss-classification error is selected to be the best classifier for that particular training set.

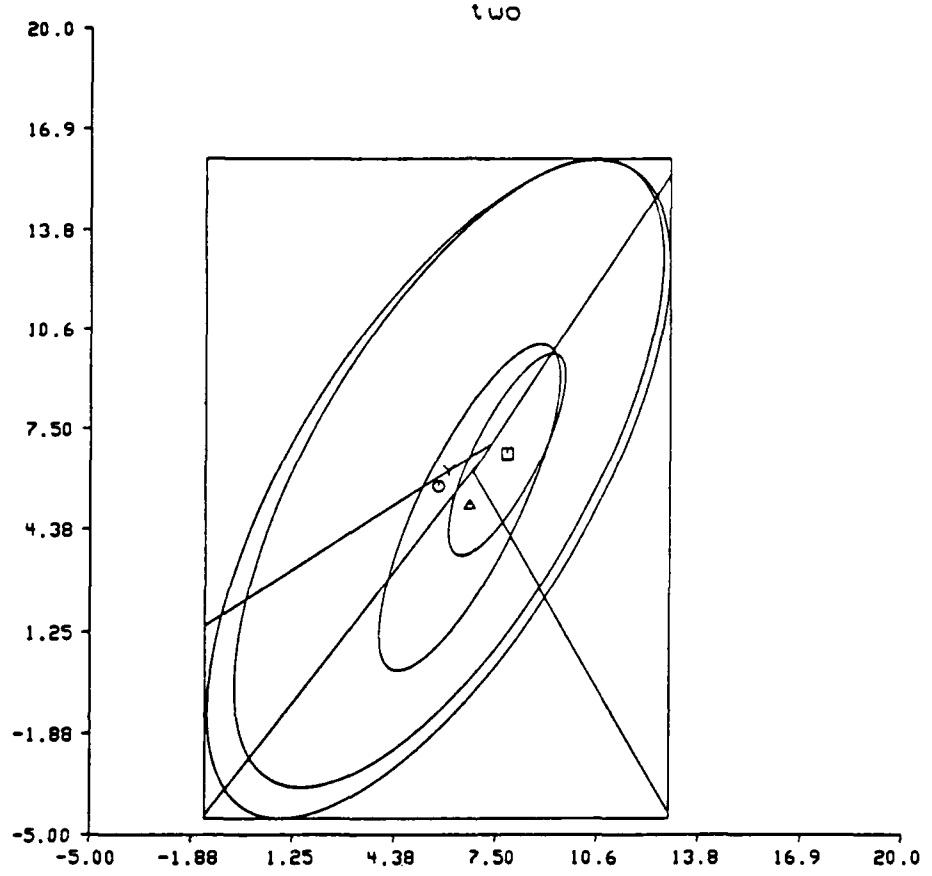


Figure 16: The *two* data set characteristic ellipses and the decision boundary.

appropriate criterion. A common criterion is to find the classifier that separates the distance between two classes. The Fisher class separability given below would measure the class sparability and is can be deduced from the linear classifiers:

$$f(\eta_i, \eta_j, \sigma_i^2, \sigma_j^2) = \frac{(\eta_i - \eta_j)^2}{\sigma_i^2 + \sigma_j^2}, \quad i \neq j. \tag{29}$$

This criterion characterizes the separation between any two classes. Note that the criterion is always non-negative which implies that its maximum is the sum of the maximum of pair-wise maxima. The physical reasoning for this criterion can be understood from the

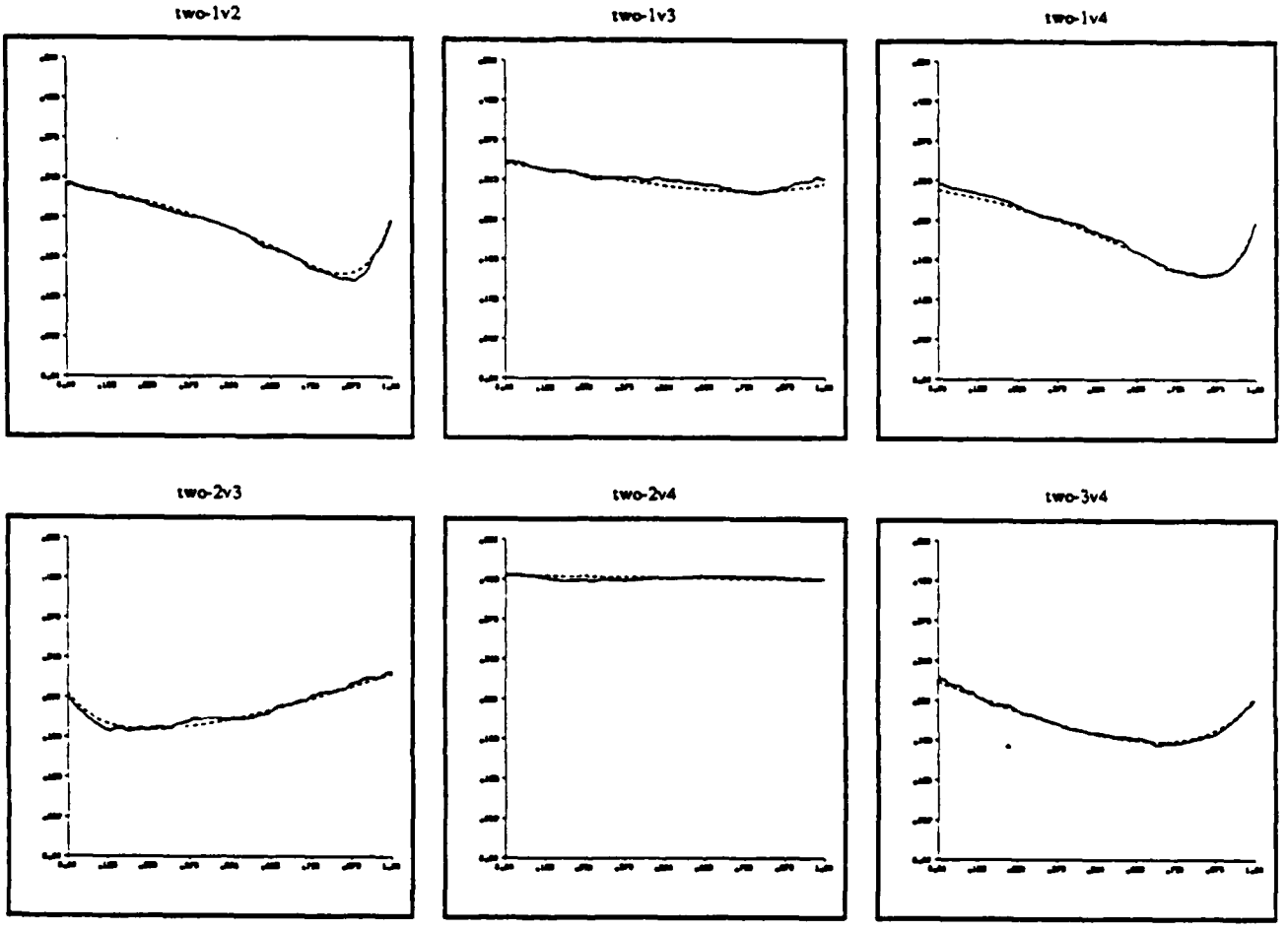


Figure 17: Pair wise class error of the *two* data set. Note that the error is higher for similar classes (i.e. classes with close statistical parameters).

numerator which is the separation between the pair-wise expected vectors. By maximizing the Fisher criterion, one clearly observes that the decision boundary is the set of points at which the separation (normalized by the variances) is a maximum. The problem can be stated as:

> *Fisher Classifier:* Find V_{ij} and V_{ijo} in equation (2) to maximize the criterion given in equation (29).

The solution for this problem can be found explicitly when the classes are normally distributed (further work is necessary to examine this problem when the distributions are

estimated). The following derivation is based on the two class problem given in [F1]. The first order necessary conditions are:

$$\frac{\partial f}{\partial V} = 0 \qquad \frac{\partial f}{\partial V_o} = 0, \tag{30}$$

by taking the partial derivatives the following equations result (recall the V is a vector and V_o is a scalar):

$$\frac{\partial f}{\partial V} = \frac{\partial f}{\partial \sigma_i^2}\frac{\partial \sigma_i^2}{\partial V} + \frac{\partial f}{\partial \sigma_j^2}\frac{\partial \sigma_j^2}{\partial V} + \frac{\partial f}{\partial \eta_i}\frac{\partial \eta_i}{\partial V} + \frac{\partial f}{\partial \eta_j}\frac{\partial \eta_j}{\partial V} \tag{31}$$

$$\frac{\partial f}{\partial V_o} = \frac{\partial f}{\partial \sigma_i^2}\frac{\partial \sigma_i^2}{\partial V_o} + \frac{\partial f}{\partial \sigma_j^2}\frac{\partial \sigma_j^2}{\partial V_o} + \frac{\partial f}{\partial \eta_i}\frac{\partial \eta_i}{\partial V_o} + \frac{\partial f}{\partial \eta_j}\frac{\partial \eta_j}{\partial V_o}. \tag{32}$$

Since the classifier is linear, (i.e. has the form $h(x) = V^T x + V_o$), then the variances and the conditional mean take the form:

$$\sigma_i^2 = V^T \Sigma_i V \tag{33}$$

$$\eta_i = V^T M_i + V_o. \tag{34}$$

Using equations (33) and (34), the first order necessary conditions become:

$$2\left[\frac{\partial f}{\partial \sigma_i^2}\Sigma_i + \frac{\partial f}{\partial \sigma_j^2}\Sigma_j\right] V = (M_i - M_j)\frac{\partial f}{\partial \eta_j}, \tag{35}$$

$$\frac{\partial f}{\partial \eta_i} + \frac{\partial f}{\partial \eta_j} = 0. \tag{36}$$

Substituting for the Fisher criterion, the following is the solution for V and V_o:

$$V = \left(\frac{1}{2}\Sigma_i + \frac{1}{2}\Sigma_j\right)^{-1}(M_i - M_j). \tag{37}$$

In comparing with the iterative linear classifier, the Fisher criterion yields maximum class separation when $s = 0.5$, hence the threshold is:

$$V_{ijo} = \frac{(M_j - M_i)^T (0.5\,\Sigma_i + 0.5\Sigma_j)^{-1} (\sigma_i^2 M_j + \sigma_j^2 M_i)}{\sigma_i^2 + \sigma_j^2}. \tag{38}$$

Equations (37) and (38) define a classifier that minimize the class separation when the data is normally distributed.

4. Classifier Trainning

We shall examine the data used to train the classifier, and study some of their properties. In particular, the statistical means and covariance matrices. One data set that will be considered is the satin weave sample with twelve drilled holes. Features are extracted from the measurement which are classified. The classifier is linear trained from the satin weave sample segmented data. Figure 19 illustrate the trainning process carried out in this chapter.

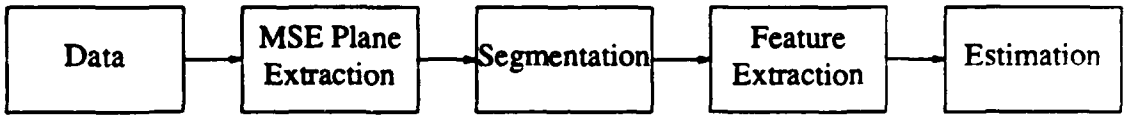


Figure 18: Classifier trainning for eddy current data.

The extracted plane stage refers to the MSE plane subtracted from the raw data in order to remove the ramp introduced by the data accquisition system. The segmentor is an auto-threshold based on the histogram of the actual measurement discribed in chapter IX as well as the MSE plane extraction. The estimation block is the estimation of the classifier parameters. In this chapter only linear classifiers are investigated, and only the first and second order statistics are required for each class. Training consists of varying the classifier parameters and choosing the classifier with minimum error probability. Other classification schemes may require different parameters for the characterization of the input data.

4.1. Feature Extraction

The features extracted for classification are the: (1) real part of the measurement, (2) imaginary part of the measurement, (3) magnitude of the measurement, (4) phase of the measurement, (5) forward difference of two adjacent measurements, and (6) forward difference of magnitude and phase. The feature vector dimension can be adjusted as necessary for sufficient separability. Note that (5) and (6) gives a total of four features, corresponding to the real, imaginary, magnitude and phase. The dimension of the feature space is then equal to eight.

5. Experimental Results

In this section some experimental of some actual measurements are presented. Scatter diagrams are shown to illustrate the clustering aspect of the measurements and to assert the the statistical approach can be used to separate flaw regions from background regions. Following scatter diagrams of actual measurements, the results of the two class detection scheme is presented.

Eddy current measurements are characterized by the in-phase and quadrature components. Most of the flaw response is of higher amplitude than that of a background. For

future reference the following is the data set that have been examined:

Table 2: Actual data used in classification experiments

Directory	Description
/c/lab.data/1987/sep30/linedata	Drilled Satin Weave Sample
/c/lab.data/1987/nov20	Sample AB -- foil target
/c/lab.data/1987/dec07	Drilled Satin weave sample
/c/lab.data/1987/dec01	Sample B -- flaw and tows

The eddy current images have been included in Chapter IX and will not reproduced here. These images have been collected under different condition to reveal the variations that exist. Also in chapter IX, the support of the flaw is shown following the segmentation.

5.1. Scatter Diagrams

The laboratory data showed some clustering. By a *cluster* we mean a collection of a close measurement points in the feature space. Thus the features are very crucial for separating the flaws and background. Figure 19 and 20 show the scatter diagram of the raw data and the filtered data resulting from subtracting the MSE plane. The horizontal axis is the real part of the measurement and the vertical axis is the imaginary part. The scatter diagram is constructed by marking the coordinates that correspond to the real and imaginary part. The Y symbol denotes a flaw measurement as identified from the segmentation and the + symbol denotes a background measurement.

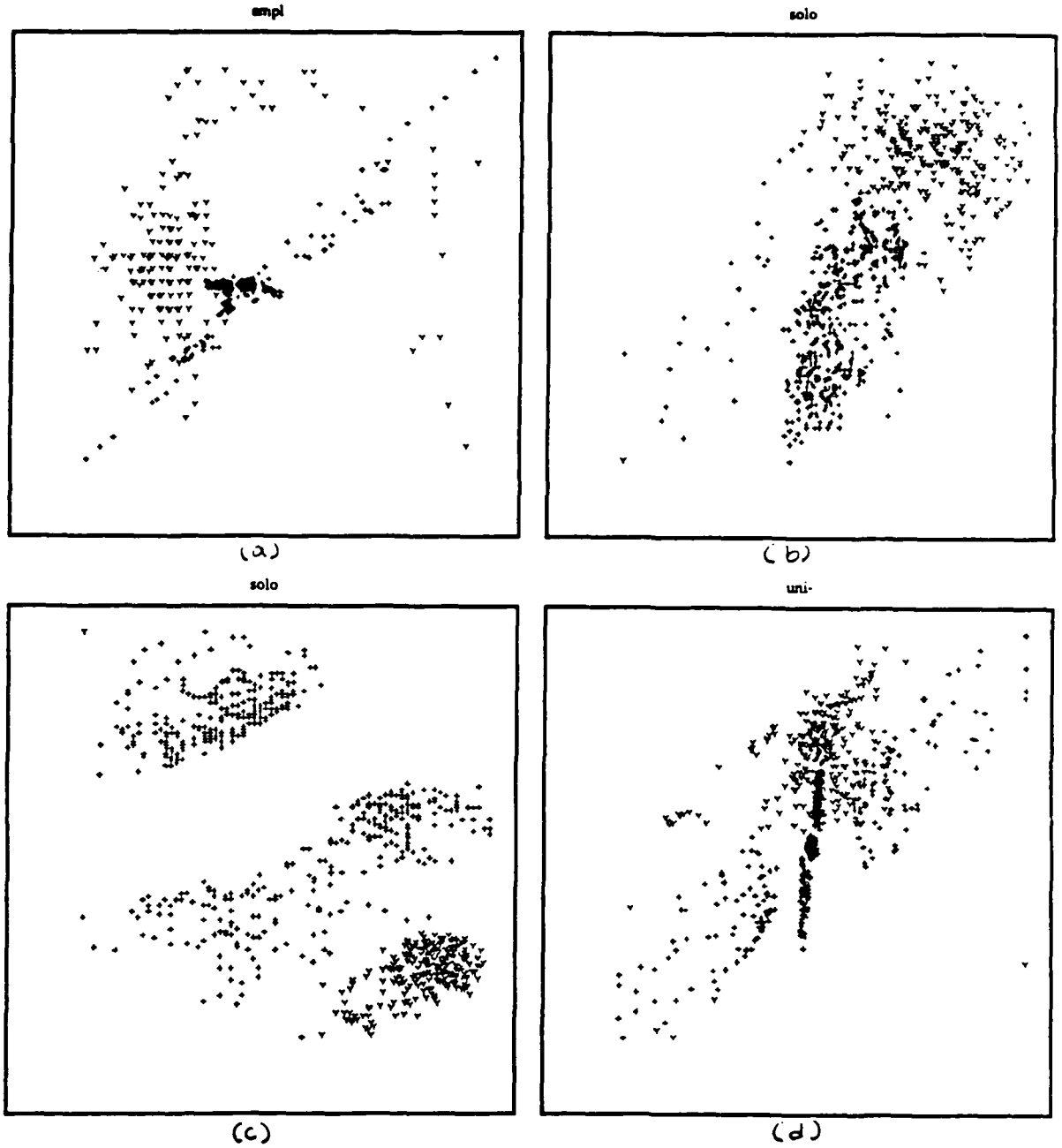


Figure 19: Scatter diagrams of the input measurements of Table 2. The horizantal axis is the real part while the vertical axis is the imaginary part (background = +, object = Y). (a) Drilled satin weave (linedata), (b) Foil target, (c) Sample B, and (d) Drilled satin weave (dec07).

In Figure 19 there are three clusters and when compared with the images, it appears that the clusters are due to flaws, background and tows. Extracting the MSE plane from the raw data, resulted in further separation of the classes as Figure 20 illustrates,

The scatter diagrams of the feature vector were shown in the SIXTH QUARTERLY REPORT and will not be reproduced here. At any rate, the depicted scatter plots do not

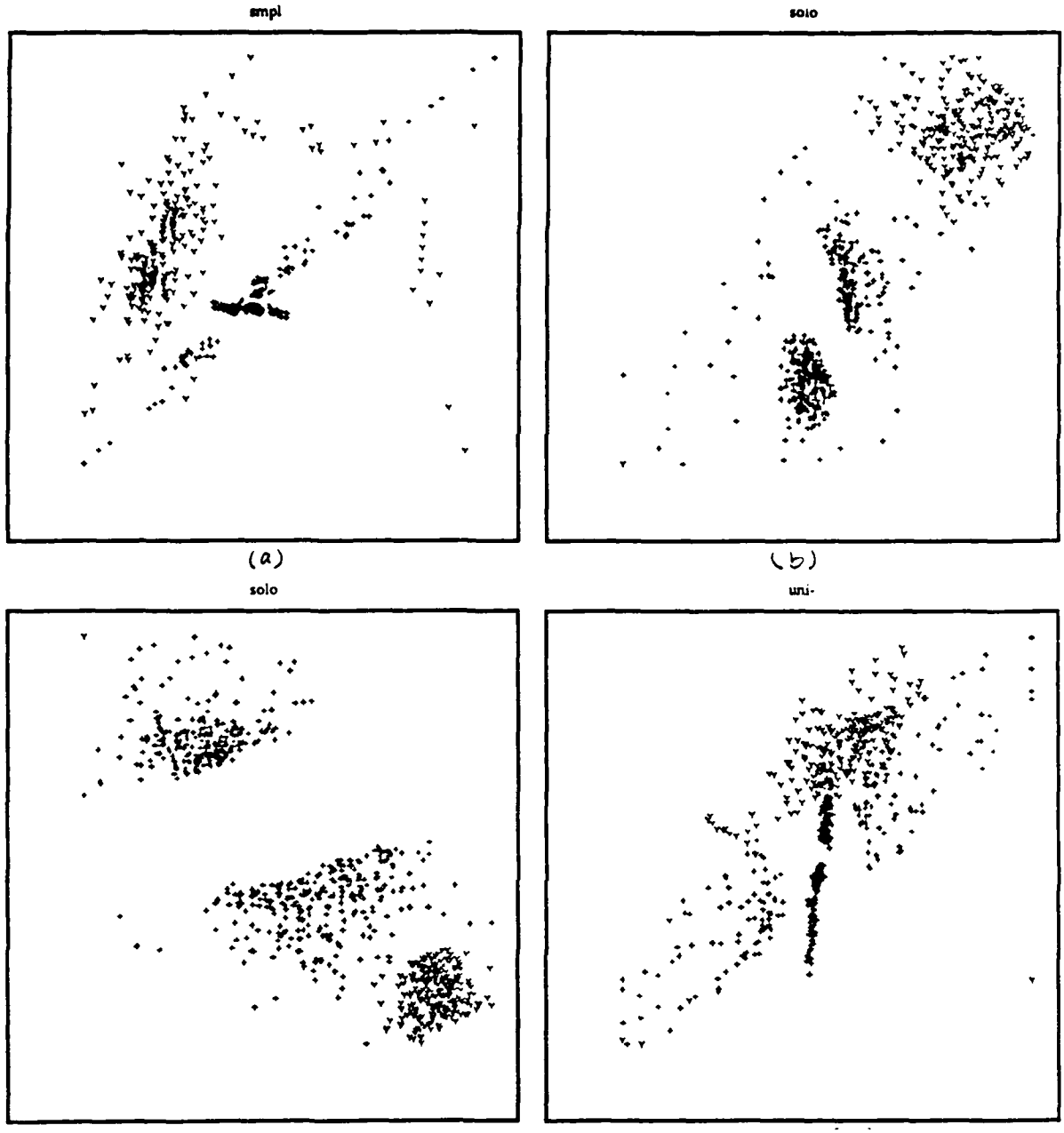


Figure 20: Scatter diagrams of the data following MSE extraction for the data set of Table 2. Note the *increase* in class separability from Figure 19. (a) Drilled satin weave (linedata), (b) Foil target, (c) Sample B, and (d) Drilled satin weave (dec07).

reflect the clustering between the eight-dimensional feature vector since only two features are shown. Figures 22, 22, 23, and 24 are the result of applying the linear classifier to the eddy current measurements.

```
PATH: /c/shamee/data/c/lab.data/1987/sep30/linedata
Data file name = smplflt1.dat   Param names = smplflt1.prm
Length = 8804                   Dimension = 8
Data file name = smplflt2.dat   Param names = smplflt2.prm
Length = 48821                  Dimension = 8

Class 1
Estimated mean
   38.687    20.962   130.762     1.879     0.002     0.003     0.004     0.000

Estimated covariance
21749.879 15774.196  9471.041  -144.701 11977.741  9192.610  5447.168   -88.188
15774.196 12523.123  6190.601  -119.397  8999.604  7387.809  3825.500   -70.786
 9471.041  6190.601 19110.424   -17.294  1795.922  1482.286  6713.511     6.353
 -144.701  -119.397   -17.294     2.886   -91.729   -76.285   -26.561     1.777
11977.741  8999.604  1795.922   -91.729 23955.611 18192.328  7243.435  -179.911
 9192.610  7387.809  1482.286   -76.285 18192.328 14775.635  5308.136  -147.067
 5447.168  3825.500  6713.511   -26.561  7243.435  5308.136 13427.813   -20.191
  -88.188   -70.786     6.353     1.777  -179.911  -147.067   -20.191     3.554

Class 2
Estimated mean
   -6.977    -3.780    28.681     3.096    -0.000     0.000    -0.000     0.000

Estimated covariance
 1085.493   584.939   342.612   -18.803   560.125   410.775   358.693   -10.943
  584.939   643.577   228.059   -22.530   386.641   336.549   273.308    -8.669
  342.612   228.059   969.428    -6.364   341.215   274.774   484.018    -7.720
  -18.803   -22.530    -6.364     2.196    -9.851    -8.537    -5.703     0.984
  560.125   386.641   341.215    -9.851  1120.193   797.395   699.912   -20.794
  410.775   336.549   274.774    -8.537   797.395   673.047   548.088   -17.207
  358.693   273.308   484.018    -5.703   699.912   548.088   967.978   -13.423
  -10.943    -8.669    -7.720     0.984   -20.794   -17.207   -13.423     1.969

s=.078125 Minimum error=.174233    Thd=1.94354

Coefficient Vector
   -0.015     0.035    -0.055     0.885    -0.001    -0.011     0.029    -0.480
```

```
PATH: /c/shamee/data/c/lab.data/1987/nov20
Data file name = soloflt1.dat
Param names = soloflt1.prm
Length = 15969
Dimension = 8
Data file name = soloflt2.dat
Param names = soloflt2.prm
Length = 41656
Dimension = 8
Class 1
Estimated mean
    5.117    -1.099   104.297     2.802    -0.002    -0.001     0.002     0.000

Estimated covariance
11519.121  2745.025  -845.662   -50.461  3972.548  1026.530  -337.464   -23.702
 2745.025  5378.015  -256.447   -83.743   958.207  2020.428   199.615   -30.742
 -845.662  -256.447  6046.718    -3.677  -215.589    55.416  1310.276    -3.099
  -50.461   -83.743    -3.677     3.948   -16.343   -28.327    -3.099     1.828
 3972.548   958.207  -215.589   -16.343  7945.003  1984.697  -553.161   -40.042
 1026.530  2020.428    55.416   -28.327  1984.697  4040.875   254.946   -59.068
 -337.464   199.615  1310.276    -3.099  -553.161   254.946  2620.884    -6.175
  -23.702   -30.742    -3.099     1.828   -40.042   -59.068    -6.175     3.656

Class 2
Estimated mean
   -1.962     0.421    63.693     2.867    -0.000     0.000    -0.000    -0.000

Estimated covariance
 4078.428  1086.503   106.316   -26.566   828.565   238.781   140.770    -6.630
 1086.503  2701.997   -67.722   -57.104   170.032   669.888     7.858   -12.424
  106.316   -67.722  2727.701    -3.134    82.635   -55.111   554.882    -1.493
  -26.566   -57.104    -3.134     3.137    -3.710   -12.941    -1.763     1.300
  828.565   170.032    82.635    -3.710  1657.118   408.813   223.404   -10.341
  238.781   669.888   -55.111   -12.941   408.813  1339.790   -47.254   -25.365
  140.770     7.858   554.882    -1.763   223.404   -47.254  1109.756    -3.256
   -6.630   -12.424    -1.493     1.300   -10.341   -25.365    -3.256     2.599

s=0  Minimum error=.351794    Thd=1.66288

Coefficient Vector
   -0.002     0.002    -0.016     0.024     0.001    -0.001     0.008    -0.017
```

```
PATH: /c/shamee/data/c/lab.data/1987/dec01
Data file name = soloflt1.dat
Param names = soloflt1.prm
Length = 15932
Dimension = 8
Data file name = soloflt2.dat
Param names = soloflt2.prm
Length = 29774
Dimension = 8
Class 1
Estimated mean
    5.033    12.785    96.185     3.238    -0.001    -0.002     0.002     0.000

Estimated covariance
 3559.699   173.638   483.715    -3.582  1046.803   217.549   126.875    -0.781
  173.638  8189.015  1248.651  -124.862    49.574  2496.769   135.175   -41.761
  483.715  1248.651  2685.957   -32.794    55.162    40.872   766.378    -7.111
   -3.582  -124.862   -32.794     3.025     0.296   -39.612    -9.797     1.063
 1046.803    49.574    55.162     0.296  2093.591   267.081   182.003    -0.482
  217.549  2496.769    40.872   -39.612   267.081  4993.419   175.950   -81.367
  126.875   135.175   766.378    -9.797   182.003   175.950  1533.089   -16.884
   -0.781   -41.761    -7.111     1.063    -0.482   -81.367   -16.884     2.127

Class 2
Estimated mean
    0.543    -4.662    47.613     3.295     0.000     0.001     0.002     0.000

Estimated covariance
 1066.634  -454.775   168.969    16.383   233.839    39.080    24.397     1.317
 -454.775  2586.739  -881.174   -56.091  -193.005   568.147  -210.253   -12.389
  168.969  -881.174  1408.470    12.241    93.257  -252.794   381.673     4.523
   16.383   -56.091    12.241     3.125     4.812   -10.728     1.357     1.398
  233.839  -193.005    93.257     4.812   467.675  -153.939   117.660     6.129
   39.080   568.147  -252.794   -10.728  -153.939  1136.191  -463.057   -23.115
   24.397  -210.253   381.673     1.357   117.660  -463.057   763.419     5.886
    1.317   -12.389     4.523     1.398     6.129   -23.115     5.886     2.796

s=.09375  Minimum error=.246675    Thd=5.08168

Coefficient Vector
   -0.001    -0.024    -0.045    -0.385    -0.001     0.012     0.023     0.211
```

```
PATH: /c/shamee/data/c/lab.data/1987/dec07
Data file name = uni-flt1.dat
Param names = uni-flt1.prm
Length = 5349
Dimension = 8
Data file name = uni-flt2.dat
Param names = uni-flt2.prm
Length = 52276
Dimension = 8
Class 1
Estimated mean
   11.047    50.170   695.336     2.828    -0.018    -0.056     0.058     0.001

Estimated covariance
164196.344 105066.508 15490.031  -212.405 101711.781 57900.414  7199.660  -134.742
105066.508 527097.063 27585.699  -920.148 72276.219 333519.156 49115.922  -693.451
15490.031 27585.699 210439.063   -25.729  5081.142 -22784.801 76739.766    48.873
 -212.405  -920.148   -25.729     2.773  -124.618  -618.350   -77.111     2.025
101711.781 72276.219  5081.142  -124.618 203421.500 130169.820 12274.604  -259.323
57900.414 333519.156 -22784.801  -618.350 130169.820 667014.750 26312.822 -1311.670
 7199.660 49115.922 76739.766   -77.111 12274.604 26312.822 153542.281   -27.757
 -134.742  -693.451    48.873     2.025  -259.323 -1311.670   -27.757     4.051

Class 2
Estimated mean
   -1.130    -5.133    55.962     3.223     0.000    -0.000     0.000     0.000

Estimated covariance
 2053.746   473.728    34.075    -2.690  1249.477   202.496  -203.647    -1.452
  473.728  5279.018  -732.142   -68.939   391.443  3461.592  -469.196   -33.194
   34.075  -732.142  4228.723     1.561   361.613  -444.336  1676.725    -2.470
   -2.690   -68.939     1.561     2.908     0.599   -33.456     5.669     1.476
 1249.477   391.443   361.613     0.599  2498.879   593.939   157.966    -0.853
  202.496  3461.592  -444.336   -33.456   593.939  6923.043  -913.533   -66.650
 -203.647  -469.196  1676.725     5.669   157.966  -913.533  3353.381     3.198
   -1.452   -33.194    -2.470     1.476    -0.853   -66.650     3.198     2.952

s=.0703125     Minimum error=.0599533   Thd=7.54627

Coefficient Vector
    0.003    -0.001    -0.043     0.170     0.000    -0.002     0.021    -0.199
```

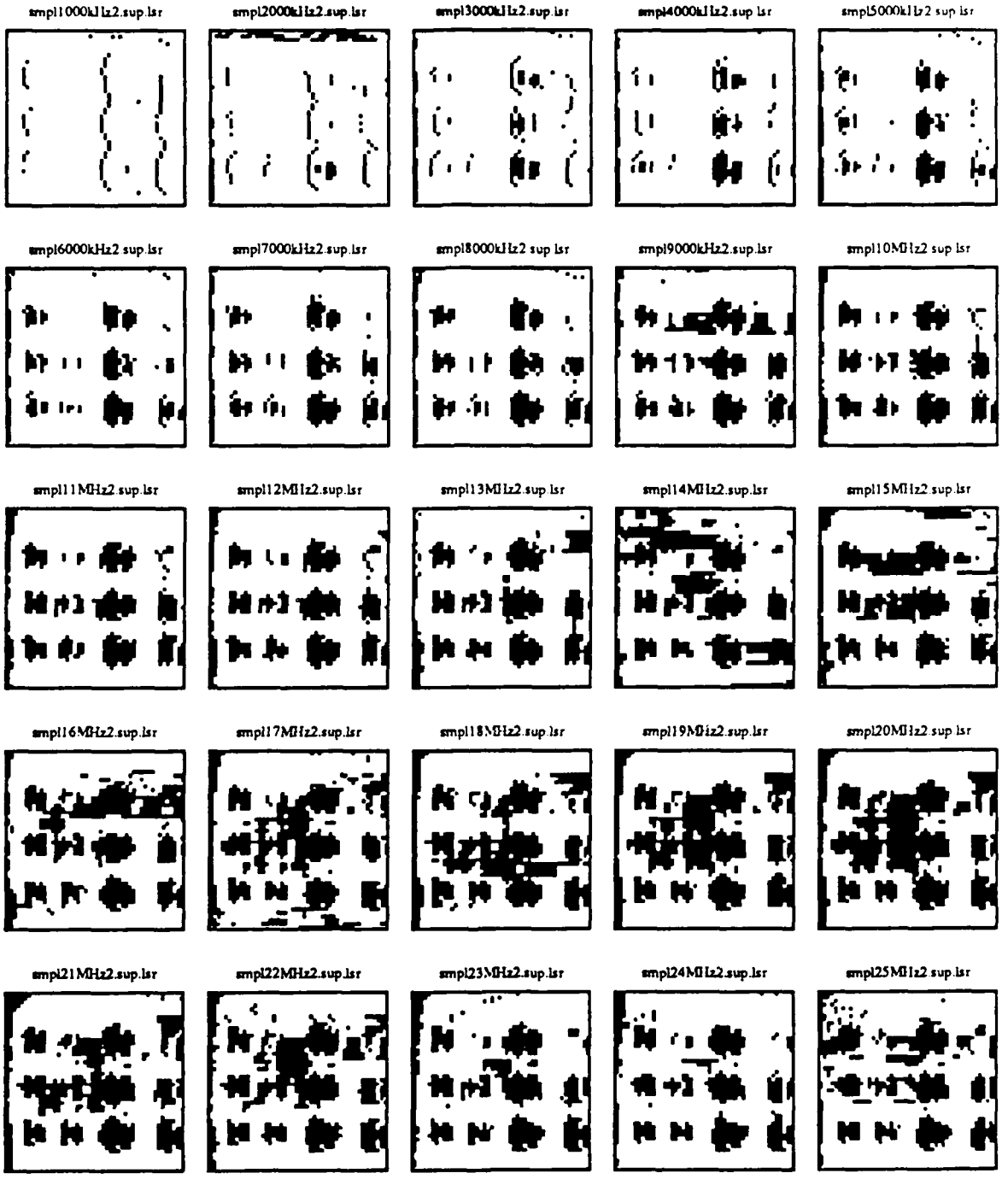


Figure 21: Linear classifier output of the sep30/linedata data set. The dark regions correspond to the support of the flaw.

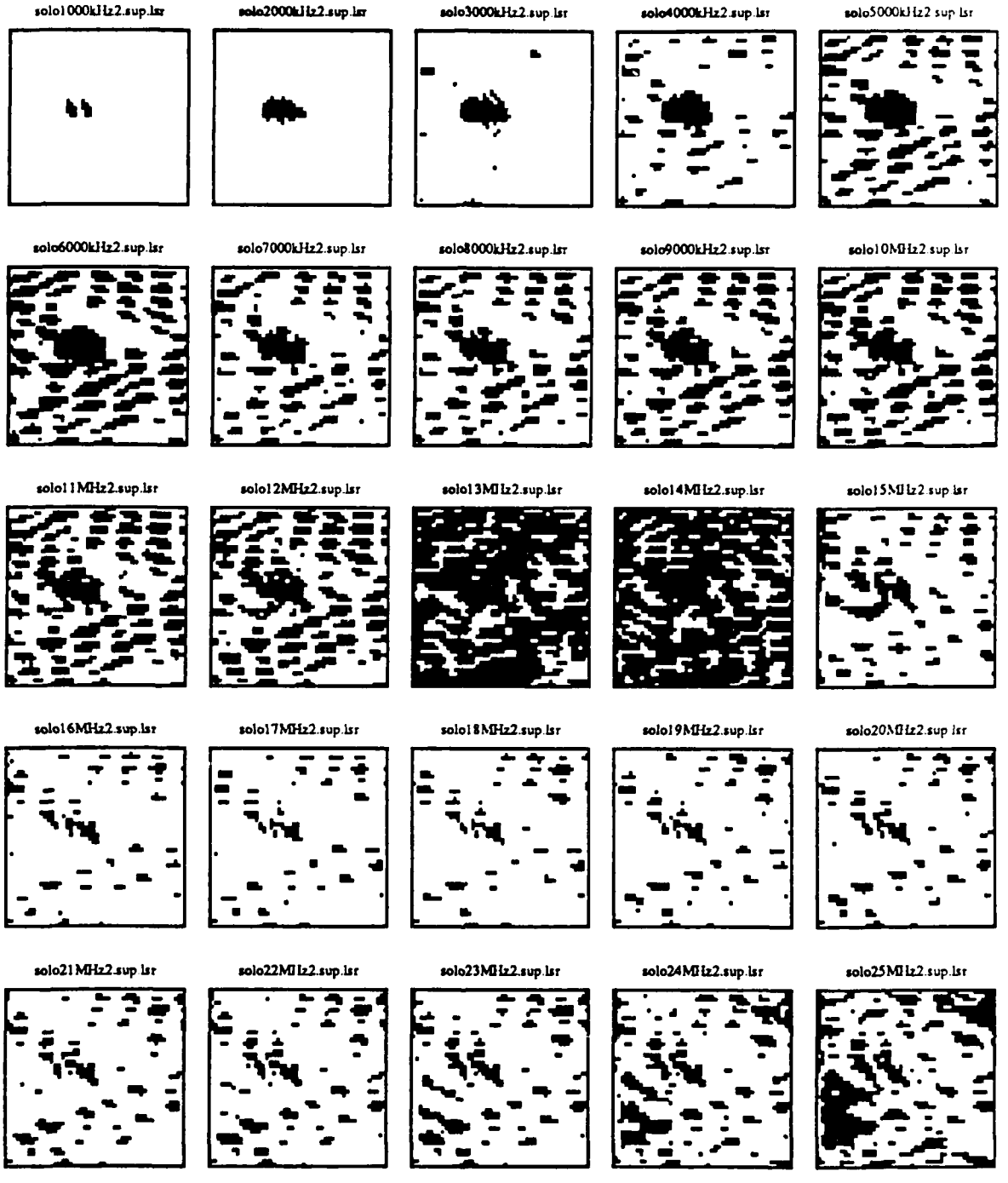


Figure 22: Linear classifier output of the nov20 data set. The dark regions correspond to the support of the flaw.

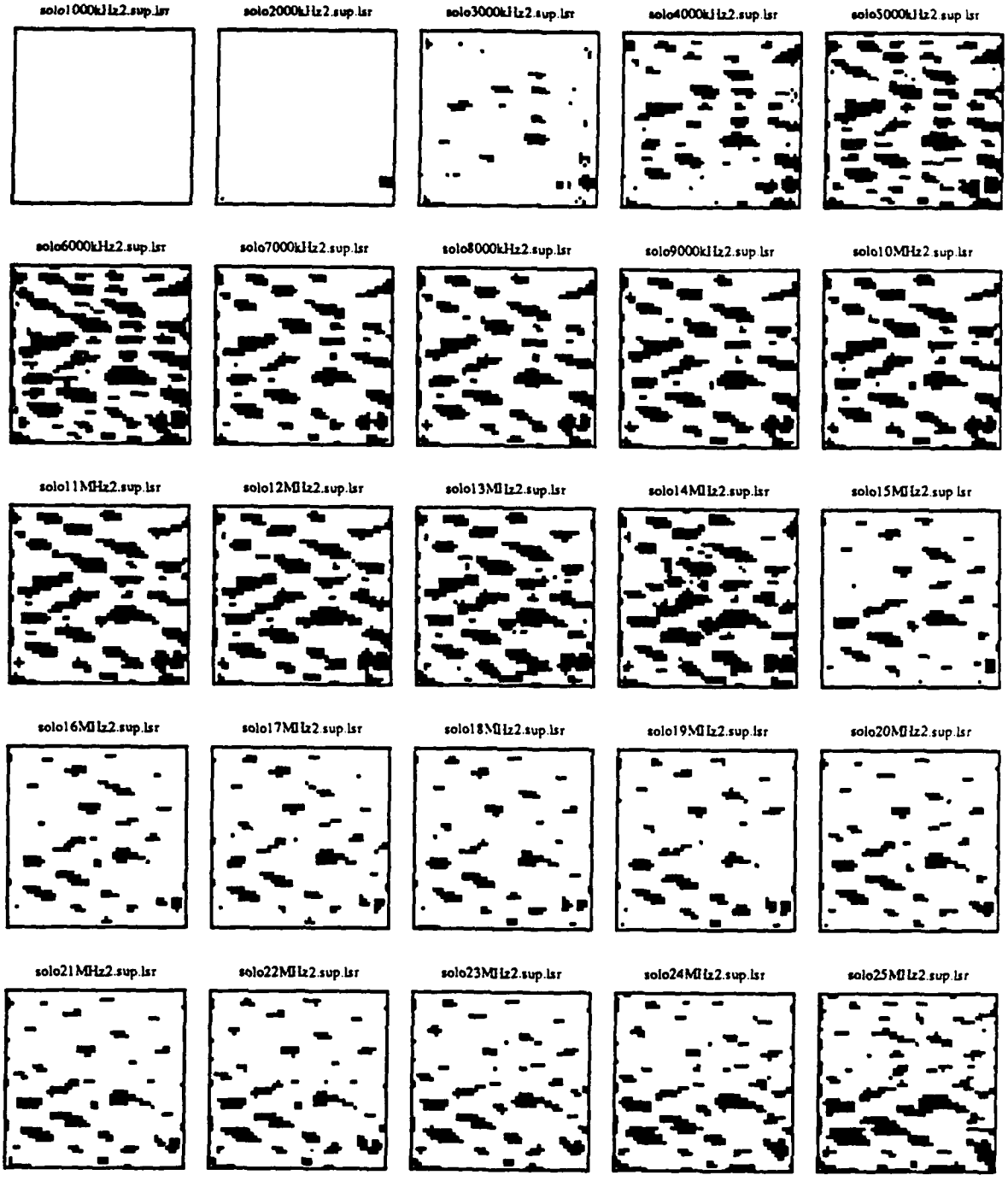


Figure 23: Linear classifier output of the dec01 data set. The dark regions correspond to the support of the flaw.

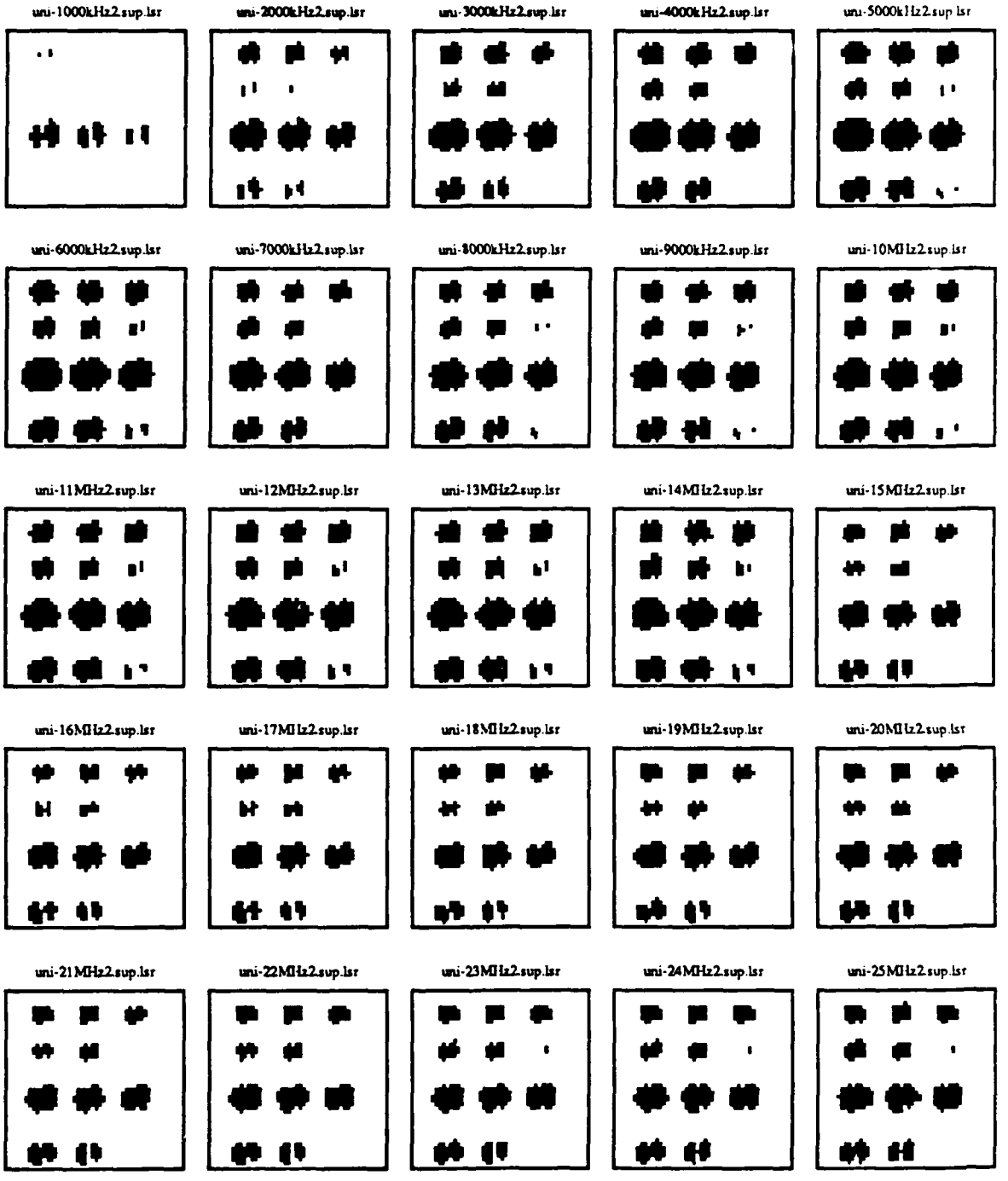


Figure 24: Linear classifier output of the dec07 data set. The dark regions correspond to the support of the flaw.

BIBLIOGRAPHY

[A1] Aho, A., Hopcroft, J., Ullman, J., *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Company, 1974.

[A2] Anderson, B., Moore, J., *Optimal Filtering*, Prentice Hall, 1979.

[F1] Fukunaga, K., *Introduction to Statistical Pattern Recognition*, Academic Press, 1972.

[G1] Gibbons, Alan *Algorithmic graph Theory*, Cambridge University Press, 1985.

[H1] Horowitz, E., Sanhi, S., *Fundamentals of Data Structures in Pascal*, Computer Science Press, 1982.

[K1] Kawata, S. and Nalcioglu, O., *Constrained Iterative Reconstruction by the Conjugate Gradient Method*, IEEE Transactions On Medical Imaging, Vol. MI-4, No. 2, June 1985, pp 65-71.

[P1] Papoulis, A., *Signal Analysis*, McGraw-Hill, 1977.

[P2] Papoulis, A., *Probability, Random Variables and Stochastic Processes*, McGraw-Hill, 1981.

[W1] Wong, E. Hajeck, B., *Stochastic Processes in Engineering Systems*, Springer-Verlag, 1981.

CHAPTER VII

A PARALLEL MACHINE FOR CONJUGATE GRADIENT INVERSION AND ITERATIVE ALGORITHMS: ARCHITECTURE

Bishara Shamee

1. Overview

The conjugate gradient method provides the basis for the present eddy current inversion schemes. The purpose of this chapter is to:

(1) Investigate the possible parallel implementation of the algorithm and devise an efficient scheme for inversion.

(2) Use the characteristics of the inversion algorithm to design a specific machine.

We applied both (1) and (2) to the conjugate gradient inversion and designed a parallel machine to invert eddy current data. We approached the design from a practical point of view, and discuss the factors that led to the architecture of the parallel machine. In addition, we estimated the inversion time to be comparable to high speed main-frames. We shall give a complete discussion of our approach and justify our assumptions.

Starting with the basic inversion algorithm, one observes that the algorithm is sequential. Sequential algorithms are those algorithms that proceed from one step to the next in order. Ordering the execution steps implies that the overall performance depends on the time and space complexity of each step. Furthermore, each step usually provides input for its successor, which prevents the execution of two or more sequential steps simultaneously. This imposes a natural limit on the time complexity. The overall time complexity can be improved only if the time complexity of each step is minimized, or in our case optimized. The optimization is thus restricted to each step of the sequential process which can be performed by parallelizing that particular step. The architecture proposed depends, in general, on the algorithm being implemented. The inversion algorithm via conjugate gradient is essentially sequential. If an alternate non-sequential algorithm inverts the data, the proposed architecture may or may not be efficient.

The machine architecture is a pipeline architecture. It achieves higher performance measure when more than one data set is inverted. The sequential order of the inversion scheme restricts the number of active elements in the pipe for a single problem. When more than one inversion problem enter the pipe, then more than one element could be active to improve the overall performance of the system. We will address performance in a later section.

The next sections will cover two inversion schemes and their corresponding architecture. Even though both schemes are based on conjugate gradients, and are used for inversion, they are different from an algorithmic point of view. We will present the algorithms, their corresponding architecture, evaluation and methods to simulate their performance on the parallel machine.

2. Inversion of Eddy Current Data Via Conjugate Gradient

The conjugate gradient is the main inversion tool used to obtain three-dimensional conductivity structure of the scanned material. Two schemes are currently used: (1) unconstrained inversion, and (2) constrained inversion. The factors that will determine the performance of the inversion are the size of memory and task partition. The first involves the space complexity of a given inversion scheme, while the second involves the dynamics of the inversion. Both aspects will be studied in order to design a parallel machine that will execute the inversion with good performance. The next two sections will address the size of the inversion problem.

2.1. Unconstrained Inversion

In the unconstrained case, the inversion is a sequential process based on the conjugate gradient method. The algorithm to invert the eddy current images is presented next without detailed explanation since the details have been presented in chapter II.

Unconstrained Conjugate Gradient Inversion

$$S_k = A \circ P_k \tag{1}$$

$$a_k = \frac{\| Q_{k-1} \|^2}{\| S_k \|^2} \tag{2}$$

$$X_k = X_{k-1} + a_k P_k \tag{3}$$

$$R_k = R_{k-1} - a_k S_k \tag{4}$$

$$Q_k = A^* \circ R_k \tag{5}$$

$$b_k = \frac{\| Q_k \|^2}{\| Q_{k-1} \|^2} \tag{6}$$

$$P_{k+1} = Q_k + b_k P_k \tag{7}$$

The following is a discussion of the unconstrained inversion scheme given in Equations (1) through (7). The known parameters prior to executing the algorithm are:

Table 1: Apriori Parameters of the Conjugate Gradient Inversion

Known Quantities, Unconstrained Case	
N_f	Number of frequencies used in the inversion.
N_x	Number of rows in the measurement.
N_y	Number of columns in the measurement.
N_z	Number of layers to be inverted.
A	A matrix with pre-computed elements.
A^*	A matrix with pre-computed elements, or derived from A.
X_o	Initial guess or a convenient starting vector.

These quantities start the algorithm and invert the data. If X_o denotes the initial guess of the conductivity, then X_o is a vector with N_z components. Each i–th component of X_o is a matrix reflecting the conductivity at the i–th layer. The jk–th element of the i–*th* matrix is the conductivity of the jk–th point at the i–th layer. Strictly speaking, the initial guess X_o and the solution X, are matrices with dimension equal to $(2N_x N_z) \times 2N_y$, where $(2N_x N_z)$ is the number of rows and $2N_y$ is the number of columns. However, we shall refer to X as a vector or as a matrix depending on the context. A typical conductivity vector X has the form shown in

Figure 1, and each iterate will have the same basic form.

$$X = \begin{bmatrix} \begin{bmatrix} \sigma_{1,1} & \sigma_{1,2} \cdots & \sigma_{1,N_y} & 0_{1,N_y+1} & 0_{1,N_y+2} \cdots & 0_{1,2N_y} \\ \sigma_{2,1} & \sigma_{2,2} \cdots & \sigma_{2,N_y} & 0_{2,N_y+1} & 0_{2,N_y+2} \cdots & 0_{2,2N_y} \\ \sigma_{N_x,1} & \sigma_{N_x,2} \cdots & \sigma_{N_x,N_y} & 0_{N_x,N_y+1} & 0_{N_x,N_y+2} \cdots & 0_{N_x,2N_y} \\ 0_{N_x+1,1} & 0_{N_x+1,2} \cdots & 0_{N_x+1,N_y} & 0_{N_x+1,N_y+1} & 0_{N_x+1,N_y+2} \cdots & 0_{N_x+1,2N_y} \\ 0_{2N_x,1} & 0_{2N_x,2} \cdots & 0_{2N_x,N_y} & 0_{2N_x,N_y+1} & 0_{2N_x,N_y+2} \cdots & 0_{2N_x,2N_y} \end{bmatrix}_1 \\ \vdots \\ \begin{bmatrix} \sigma_{1,1} & \sigma_{1,2} \cdots & \sigma_{1,N_y} & 0_{1,N_y+1} & 0_{1,N_y+2} \cdots & 0_{1,2N_y} \\ \sigma_{2,1} & \sigma_{2,2} \cdots & \sigma_{2,N_y} & 0_{2,N_y+1} & 0_{2,N_y+2} \cdots & 0_{2,2N_y} \\ \sigma_{N_x,1} & \sigma_{N_x,2} \cdots & \sigma_{N_x,N_y} & 0_{N_x,N_y+1} & 0_{N_x,N_y+2} \cdots & 0_{N_x,2N_y} \\ 0_{N_x+1,1} & 0_{N_x+1,2} \cdots & 0_{N_x+1,N_y} & 0_{N_x+1,N_y+1} & 0_{N_x+1,N_y+2} \cdots & 0_{N_x+1,2N_y} \\ 0_{2N_x,1} & 0_{2N_x,2} \cdots & 0_{2N_x,N_y} & 0_{2N_x,N_y+1} & 0_{2N_x,N_y+2} \cdots & 0_{2N_x,2N_y} \end{bmatrix}_{N_z} \end{bmatrix},$$

Figure 1: The conductivity matrix used in the Conjugate Gradient Inversion.

The conductivity at the *jk*-th sample point is denoted by σ_{jk}, and the layer number has to specified in order to uniquely identify the conductivity. The 0's have been appended in order to use an efficient scheme for convolution and it is not necessary to store these elements. If the measured data is denoted by Y, then this vector would have dimension equal to $2 N_f N_x \times N_y$ because Y has a real (in-phase) and imaginary (quadrature) components for each frequency. The elements of A are known a priori reflecting the sensor model and other parameters. The first step in the inversion is to evaluate the quantity R_o given by:

$$R_o = Y - A \bigcirc X_o, \tag{8}$$

where the *circle* operation $\bigcirc$ resembles matrix multiplication to a great extent. Equation (9) defines the circle operation to mean a sequence of elementary operations on two matrices as follows: Suppose A is an augmented block matrix composed of submatrices T_{ij}, then $A \bigcirc X$ is a vector whose i-th component is a matrix *defined* by:

$$(A \bigcirc X)_i = \sum_{j=1} T_{ij} * X_j = T_{i1} * X_1 + T_{i2} * X_2 + \cdots T_{iN_z} * X_{N_z}, \qquad i = 1, 2, \ldots, 2 N_f, \tag{9}$$

where * is the two dimensional convolution. If the convolutions were replaced by the matrix product, then $\bigcirc$ would reduce to ordinary matrix product. It is not our intent to construct a mathematical structure showing the relation of $\bigcirc$ to ordinary matrix product, however, it is important to note the similarity with the ordinary matrix product, since if the convolution is defined as a fundamental operation (in terms of hardware as multiplication), then many of the available results about matrix product optimization extend in a natural way to the operation defined in Equation (2). The problem of reducing the inversion time would then focus on the enhancement of the convolution. The structure of the matrix A is shown below:

$$A = \begin{bmatrix} T_{1,1} & T_{2,1} & \cdots & T_{1,N_z} \\ T_{2,1} & T_{2,2} & \cdots & T_{2,N_z} \\ T_{2N_f,1} & T_{2N_f,2} & \cdots & T_{2N_f,N_z} \end{bmatrix}.$$

Expanding each T_{ij} yields the following equivalent: the following is equivalent:

$$A = \begin{bmatrix} \begin{bmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,2N_y} \\ t_{2,1} & t_{2,2} & \cdots & t_{,2,2N_y} \\ t_{2N_x,1} & t_{2N_x,2} & \cdots & t_{2N_x,2N_y} \end{bmatrix}_{1,1} & \begin{bmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,2N_y} \\ t_{2,1} & t_{2,2} & \cdots & t_{,2,2N_y} \\ t_{2N_x,1} & t_{2N_x,2} & \cdots & t_{2N_x,2N_y} \end{bmatrix}_{1,2} & \cdots & \begin{bmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,2N_y} \\ t_{2,1} & t_{2,2} & \cdots & t_{,2,2N_y} \\ t_{2N_x,1} & t_{2N_x,2} & \cdots & t_{2N_x,2N_y} \end{bmatrix}_{1,N_Z} \\ \begin{bmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,2N_y} \\ t_{2,1} & t_{2,2} & \cdots & t_{,2,2N_y} \\ t_{2N_x,1} & t_{2N_x,2} & \cdots & t_{2N_x,2N_y} \end{bmatrix}_{2,1} & \begin{bmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,2N_y} \\ t_{2,1} & t_{2,2} & \cdots & t_{,2,2N_y} \\ t_{2N_x,1} & t_{2N_x,2} & \cdots & t_{2N_x,2N_y} \end{bmatrix}_{2,2} & \cdots & \begin{bmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,2N_y} \\ t_{2,1} & t_{2,2} & \cdots & t_{,2,2N_y} \\ t_{2N_x,1} & t_{2N_x,2} & \cdots & t_{2N_x,2N_y} \end{bmatrix}_{2,N_Z} \\ \begin{bmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,2N_y} \\ t_{2,1} & t_{2,2} & \cdots & t_{,2,2N_y} \\ t_{2N_x,1} & t_{2N_x,2} & \cdots & t_{2N_x,2N_y} \end{bmatrix}_{2N_f,1} & \begin{bmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,2N_y} \\ t_{2,1} & t_{2,2} & \cdots & t_{,2,2N_y} \\ t_{2N_x,1} & t_{2N_x,2} & \cdots & t_{2N_x,2N_y} \end{bmatrix}_{2N_f,2} & \cdots & \begin{bmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,2N_y} \\ t_{2,1} & t_{2,2} & \cdots & t_{,2,2N_y} \\ t_{2N_x,1} & t_{2N_x,2} & \cdots & t_{2N_x,2N_y} \end{bmatrix}_{2N_f,N_Z} \end{bmatrix}.$$

Figure 2: Matrix A used in Conjugate Gradient Inversion.

The dimension of the A matrix is $(2N_f)\,(2N_x) \times (2N_y)\,N_z$ (i.e. A has $2N_f\,2N_x$ rows and $2N_y\,N_z$ columns).

The measurement vector Y has the dimensions of $(2N_f)\,N_x \times N_y$. Note that with dimension of Y, there is a discrepancy in Equation (2), since necessarily the i–th component of $A \bigcirc X$ is $2N_x \times 2N_y$ which does not equal the dimension of the i–th component of Y. Deleting all entries but the first $N_x \times N_y$ of the i–th component of $A \bigcirc X$ resolves the discrepancy. It appears that there is some redundancy in this formulation, i.e. augment additional entries to X and then discard elements from $A \bigcirc X$. Some analysis may be necessary to investigate the current formulation and/or alternatives (if possible) to perform efficient convolutions.

The unconstrained algorithm, depends on a matrix A^* obtained from A in the following manner. Recall, A is composed of submatrices $T_{i,j}$, $i = 1, \ldots, 2N_f$, $j = 1, \ldots, N_z$. For each submatrix $T_{i,j}$, the Hermitian transpose $T^H_{i,j}$ defines the i, j-th submatrix of A^*. In other words, the following is A^*:

$$A^* = \begin{bmatrix} T^H_{1,1} & T^H_{2,1} & \cdots & T^H_{1,N_z} \\ T^H_{2,1} & T^H_{2,2} & \cdots & T^H_{2,N_z} \\ T^H_{2N_f,1} & T^H_{2N_f,2} & \cdots & T^H_{2N_f,N_z} \end{bmatrix}.$$

For completeness, the Hermitian Transpose means to take the complex conjugate of each entry and transpose the resulting matrix. A^* and A will have different dimensions (but not size) only if the measurement dimensions are not equal (i.e. $N_x \neq N_y$). When A^* is expanded, the following is equivalent (in terms of the elements of A):

$$A^* = \begin{bmatrix} \begin{bmatrix} \overline{t}_{1,1} & \overline{t}_{2,1} & \cdots & \overline{t}_{1,2N_x} \\ \overline{t}_{1,2} & \overline{t}_{2,2} & \cdots & \overline{t}_{2,2N_x} \\ \overline{t}_{1,2N_y} & \overline{t}_{2,2N_y} & \cdots & \overline{t}_{2N_y,2N_x} \end{bmatrix}_{1,1} & \begin{bmatrix} \overline{t}_{1,1} & \overline{t}_{2,1} & \cdots & \overline{t}_{1,2N_x} \\ \overline{t}_{1,2} & \overline{t}_{2,2} & \cdots & \overline{t}_{2,2N_x} \\ \overline{t}_{1,2N_y} & \overline{t}_{2,2N_y} & \cdots & \overline{t}_{2N_y,2N_x} \end{bmatrix}_{1,2} & \cdots & \begin{bmatrix} \overline{t}_{1,1} & \overline{t}_{2,1} & \cdots & \overline{t}_{1,2N_x} \\ \overline{t}_{1,2} & \overline{t}_{2,2} & \cdots & \overline{t}_{2,2N_x} \\ \overline{t}_{1,2N_y} & \overline{t}_{2,2N_y} & \cdots & \overline{t}_{2N_y,2N_x} \end{bmatrix}_{1,N_z} \\ \begin{bmatrix} \overline{t}_{1,1} & \overline{t}_{2,1} & \cdots & \overline{t}_{1,2N_x} \\ \overline{t}_{1,2} & \overline{t}_{2,2} & \cdots & \overline{t}_{2,2N_x} \\ \overline{t}_{1,2N_y} & \overline{t}_{2,2N_y} & \cdots & \overline{t}_{2N_y,2N_x} \end{bmatrix}_{2,1} & \begin{bmatrix} \overline{t}_{1,1} & \overline{t}_{2,1} & \cdots & \overline{t}_{1,2N_x} \\ \overline{t}_{1,2} & \overline{t}_{2,2} & \cdots & \overline{t}_{2,2N_x} \\ \overline{t}_{1,2N_y} & \overline{t}_{2,2N_y} & \cdots & \overline{t}_{2N_y,2N_x} \end{bmatrix}_{2,2} & \cdots & \begin{bmatrix} \overline{t}_{1,1} & \overline{t}_{2,1} & \cdots & \overline{t}_{1,2N_x} \\ \overline{t}_{1,2} & \overline{t}_{2,2} & \cdots & \overline{t}_{2,2N_x} \\ \overline{t}_{1,2N_y} & \overline{t}_{2,2N_y} & \cdots & \overline{t}_{2N_y,2N_x} \end{bmatrix}_{2,N_z} \\ \begin{bmatrix} \overline{t}_{1,1} & \overline{t}_{2,1} & \cdots & \overline{t}_{1,2N_x} \\ \overline{t}_{1,2} & \overline{t}_{2,2} & \cdots & \overline{t}_{2,2N_x} \\ \overline{t}_{1,2N_y} & \overline{t}_{2,2N_y} & \cdots & \overline{t}_{2N_y,2N_x} \end{bmatrix}_{2N_f,1} & \begin{bmatrix} \overline{t}_{1,1} & \overline{t}_{2,1} & \cdots & \overline{t}_{1,2N_x} \\ \overline{t}_{1,2} & \overline{t}_{2,2} & \cdots & \overline{t}_{2,2N_x} \\ \overline{t}_{1,2N_y} & \overline{t}_{2,2N_y} & \cdots & \overline{t}_{2N_y,2N_x} \end{bmatrix}_{2N_f,2} & \cdots & \begin{bmatrix} \overline{t}_{1,1} & \overline{t}_{2,1} & \cdots & \overline{t}_{1,2N_x} \\ \overline{t}_{1,2} & \overline{t}_{2,2} & \cdots & \overline{t}_{2,2N_x} \\ \overline{t}_{1,2N_y} & \overline{t}_{2,2N_y} & \cdots & \overline{t}_{2N_y,2N_x} \end{bmatrix}_{2N_f,N_z} \end{bmatrix},$$

Figure 3: Matrix A^* used in Conjugate Gradient Inversion.

where $\overline{\ }$ is the complex conjugation operator. The dimension of the matrix A^* is $(2N_f)(2N_y) \times (2N_x)N_z$ (see Figure 3).

The input vector is the actual laboratory measurement obtained from scanning the workpiece and is frequency dependent. Both A and A^* also depend on the sensor since they incorporate the its model. The input measurement is denoted by Y, with dimension equal to $(2N_f)N_x \times N_y$. Y has the following form:

$$Y = \begin{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,N_y} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,N_y} \\ . & . & & . \\ b_{N_x,1} & b_{N_x,2} & \cdots & b_{N_x,N_y} \end{bmatrix}_1 \\ \\ . \\ . \\ . \\ \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,N_y} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,N_y} \\ . & . & & . \\ b_{N_x,1} & b_{N_x,2} & \cdots & b_{N_x,N_y} \end{bmatrix}_{2N_f} \end{bmatrix},$$

Figure 4: The input measurement matrix.

where $b_{i,j}$ denotes the measurement at the ij-th sample point. The frequency must also be specified to identify a particular sample point. The real (in-phase) matrices occupy the first N_f rows and the imaginary (quadrature) matrices occupy the remaining $N_f + 1$ to $2\,N_f$.

In summary, Table 2 and 3 show the equations and the memory requirements for the unconstrained inversion. The total memory requirements depends on the product of the inversion parameters given in Table 1. In the architecture proposed, the amount this memory is essential to speed the inversion. In Table 3, some numerical calculations have been performed for different parameters.

Table 2: Memory Requirements For Unconstrained Inversion

Unconstrained Inversion Space Complexity		
Variable	Equation	Size in numbers
S_k	$S_k = A \bigcirc P_k$	$2 N_f N_x \times N_y$
a_k	$a_k = \frac{\|\|Q_{k-1}\|\|^2}{\|\|S_k\|\|^2}$	1
X_k	$X_k = X_{k-1} + a_k P_k$	$2 N_f N_x \times N_y$
R_k	$R_k = R_{k-1} - b_k S_k$	$2 N_f N_x \times N_y$
Q_k	$Q_k = A^* \bigcirc P_k$	$2 N_f N_x \times N_y$
b_k	$b_k = \frac{\|\|Q_k\|\|^2}{\|\|Q_{k-1}\|\|^2}$	1
P_{k+1}	$P_{k+1} = Q_k + b_k P_k$	$2 N_f N_x \times N_y$
A		$(2 N_f)(2 N_x) \times (2 N_y) N_z$
A^*		$(2 N_f)(2 N_y) \times (2 N_x) N_z$
Total		$(2 N_f)(N_x N_y)(8 N_z + 5)$

Table 3: Unconstrained Inversion memory requirements for some configurations.

Memory Requirement				
N_s	N_f	N_x	N_y	Size in Numbers
1	1	16	16	6.5K
1	1	32	32	26.0K
1	1	64	64	104.0K
1	1	128	128	416.0K
2	2	16	16	21.0K
2	2	32	32	84.0K
2	2	64	64	336.0K
2	2	128	128	1.3M
2	4	16	16	42.0K
2	4	32	32	168.0K
2	4	64	64	672.0K
2	4	128	128	2.6M
2	8	16	16	84.0K
2	8	32	32	336.0K
2	8	64	64	1.3M
2	8	128	128	5.3M
2	16	16	16	168.0K
2	16	32	32	672.0K
2	16	64	64	2.6M
2	16	128	128	10.5M
4	8	16	16	148.0K
4	8	32	32	592.0K
4	8	64	64	2.3M
4	8	128	128	9.3M
4	16	16	16	296.0K
4	16	32	32	1.2M
4	16	64	64	4.6M
4	16	128	128	18.5M
8	8	16	16	276.0K
8	8	32	32	1.1M
8	8	64	64	4.3M
8	8	128	128	17.3M
8	16	16	16	552.0K
8	16	32	32	2.2M
8	16	64	64	8.6M
8	16	128	128	34.5M
16	16	16	16	1.0M
16	16	32	32	4.2M
16	16	64	64	16.6M
16	16	128	128	66.5M
16	25	16	16	1.6M
16	25	32	32	6.5M
16	25	64	64	26.0M
16	25	128	128	103.9M

2.2. Constrained Inversion

The constrained inversion is used when the solution vector is known *apriori* to be contained in some subset. Hence, the solution must satisfy certain inequality constraints. The constraints are used to formulate the inversion as discussed in chapter II. The space complexity is on the order of the unconstrained inversion, and the main distinction is the algorithm flow and control as illustrated below.

Constrained Inversion

Step 1:

$$R_1 = B - A\,S\,\sigma_1, \quad Q_1 = S\,A\,R_1$$

if $\|Q_1\| < \varepsilon$ Then **terminate.**

Step 2:

$$H_{ij} = \begin{cases} 1 & \sigma_{ij}\,S_{ij} \in [\sigma_{\min}, \sigma_{\max}] \\ 0 & \textit{otherwise} \end{cases}$$

If $H \equiv 0$ Then **Step 5**

$$P_1 = H\,Q_1$$

Step 3:

$$T = A\,S\,P_k, \quad a_k = \left[\frac{\|H\,Q_k\|_2}{\|T\|_2}\right]^2, \quad X_{k+1} = X_k + a_k\,P_k$$

If $\sigma_{ij} \notin (\sigma_{\min}, \sigma_{\max})$ for any i, j Then **Step 4**

$$R_{k+1} = R_k - a_k\,T, \quad Q_{k+1} = H\,S\,A^*\,R_{k+1}$$

If $\|Q_{k+1}\|_2 < \varepsilon$ or $\dfrac{\|R_{k+1}\|}{\|B\|_2} < \delta$ Then **terminate.**

$$b_k = \frac{\|Q_{k+1}\|_2^2}{\|Q_k\|^2}, \quad P_{k+1} = Q_{k+1} + b_k\,P_k, \quad \text{Go to } \textbf{Step 3}$$

Step 4:

$$c = \min\left[\min_{\sigma_{ij}\,S_{ij} > X_{\max}}\left[\frac{X_{\max} - \sigma_{ij}\,S_{ij}}{S_{ij}\,P_{ij}}\right], \min_{\sigma_{ij}\,S_{ij} < X_{\min}}\left[\frac{X_{\min} - \sigma_{ij}\,S_{ij}}{S_{ij}\,P_{ij}}\right]\right].$$

$$\sigma_1 = \sigma_{k+1} + c\,P_{k+1}$$

Step 5:

If $\|Q_{k+1}\|_2 < \varepsilon$ Then **terminate.**

$$P_{ij} = \min_{\lambda_{ij} > 0}\left[Q_{ij} - \sum d_{ij}\,\lambda_{ij}\right]. \quad \text{If } \|P\|_2 < \varepsilon \text{ Then } \textbf{terminate.}$$

$$\sigma_1 = \sigma_{k+1} + P$$

$$c = \min\left[\min_{\sigma_{ij}\,S_{ij} > X_{\max}}\left[\frac{X_{\max} - \sigma_{ij}\,S_{ij}}{S_{ij}\,P_{ij}}\right], \min_{\sigma_{ij}\,S_{ij} < X_{\min}}\left[\frac{X_{\min} - \sigma_{ij}\,S_{ij}}{S_{ij}\,P_{ij}}\right]\right].$$

$$c = \min\left[c, \frac{P\,Q}{P^*\,A^*\,A\,P} - 1\right], \quad \sigma_1 = \sigma_{k+1} + c\,P, \quad \text{Go to } \textbf{Step 1}$$

3. Pipeline Architecture For Inversion

In this section, the architecture for executing the conjugate gradient inversion is presented. The basic idea is to partition the conjugate gradient algorithm into steps and execute the steps in the order they appear. It should be pointed out again, that the CG algorithm is sequential which imposes a natural limit on the degree of parallelism that could be achieved.

In the following, a general pipeline architecture is presented. This architecture achieves parallelism within the stages of the pipe and as will be seen it achieves higher performance when more than one data set are inverted. Figure 5 below illustrates the pipeline architecture of the inversion. As the data enters the pipeline, the First In First Out (FIFO) queue organizes the computational load and keeps the pipe full, after each iterate, a decision is made whether to terminate the inversion or to enter the FIFO again. If the conjugate gradient algorithm terminates, then the solution is stored (addresses to the solution) in the Last In First Out (LIFO) queue. The incoming data can use the previous solutions as initial guesses. For neighboring scan areas, the solution vectors are in the neighborhood of each other. This observation could speed the convergence of the conjugate gradient inversion.

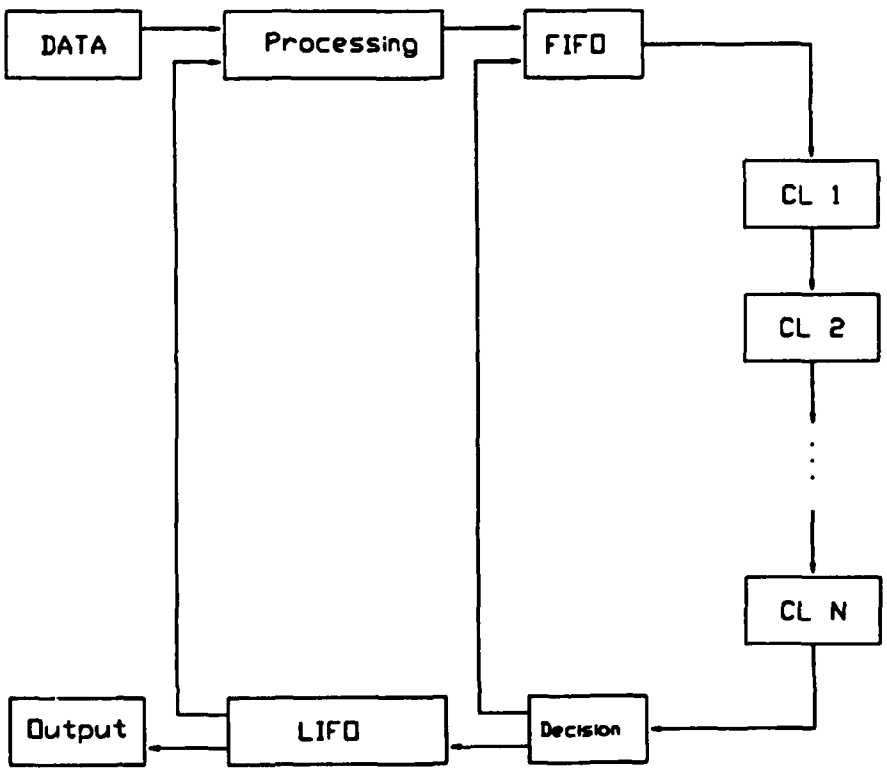


Figure 5: Pipeline architecture for conjugate gradient inversion.

Each computational layer is denoted by *CL* and is used to perform certain tasks corresponding to the Conjugate Gradient. The processing consists of the initialization of the variables and for using the data in the LIFO for initialization. To achieve high throughput, as the data is obtained over a large workpiece (or resampling the same area), the previous solution can be used to initialize the current solution and thus the speed up is obtained from the prior solutions and the pipe structure. In the following, the unconstrained graph is obtained and mapped onto the pipeline architecture, then we move to discuss the constrained case and then we present the task assignment for sufficiently large class of computational algorithms which are suitable for this pipeline architecture. Note that parallelization is not broken down to the lowest possible level, since the lowest level of parallelization increases the complexity of the design. The use of available computing VLSI chips, the system could achieve very good behavior at higher levels. From the previous section, the fundamental steps could be seen in the following Algorithm Graph:

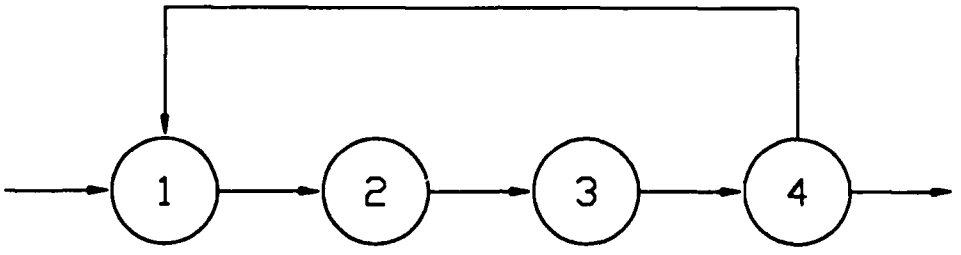


Figure 6: Algorithm Graph of the unconstrained conjugate gradient inversion. Note the similarity with the pipeline architecture from this graph.

Figure 6 illustrates the sequential process of the unconstrained version. The parallelization is performed at each of the steps in order to decrease the inversion time.

If the LIFO data is used to initialize the incoming inversion problems, then a flawed scan region may initialize an unflawed scan region. The deviation of the initial condition (from zero) may slow the convergence of the unflawed region. A method to avoid such a problem is to use the segmentor developed in Chapter IX to isolate flawed scan regions from unflawed regions. Then the solution to a previous flawed region would initialize the incoming flawed region. The increase in processing overhead introduced by segmentation must be evaluated to determine if the overall inversion time would improve.

The constrained case is different because of the algorithm control and the additional computational load to satisfy the constraints. The main difference from an algorithmic point of view, are the cycles present in the constrained case as illustrated in Figure 7. The repeated step would delay other steps present in the pipe if the cycle duration is long enough. In a later section, a general scheme to execute the cycles without delay will be discussed. For the conjugate gradient inversion, the looping in step 3 could be eliminated by examining the behavior of the constrained case if step 3 did not return to itself, rather exit to the beginning of the algorithm. It was determined that the constrained conjugate gradient will converge if step 3 was not executed repeatedly. The following graph is the

modified constrained conjugate gradient for inversion.

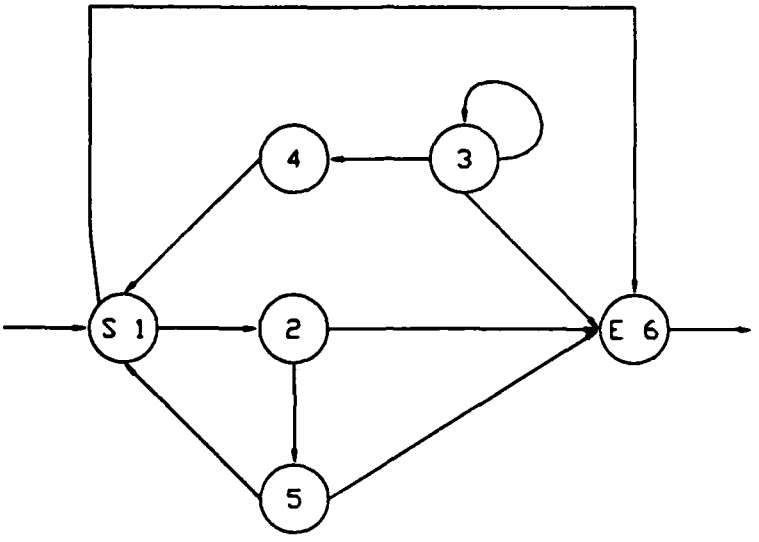


Figure 7: Algorithm graph for the constrained case. The cycle (step 3) is not desired for reasons discussed below.

Appendix A of this chapter, describes the details of the conjugate gradient modification. Next, we present the parallel machine motivated by the conjugate gradient inversion. The machine can be configured into a pipeline architecture to execute the conjugate gradient and improve the system throughput.

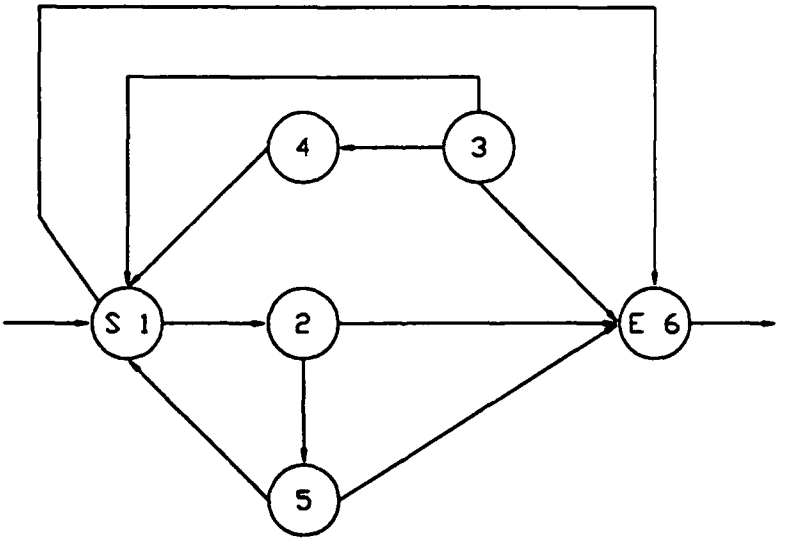


Figure 8: Algorithm graph of the constrained case after modification of the cycle.

4. A Parallel Machine for the Conjugate Gradient

In this section we present and discuss the architecture of a parallel machine to execute the conjugate gradient inversion. We shall denote our machine by the *PWP Machine*. While the PWP machine presented is not new in the domain of parallel computers, the recent development in Digital Signal Processing (DSP) chips permits the design of a relatively cheap and high performance system. The basic design philosophy of the PWP machine is to: (1) achieve parallelism at higher levels, (2) minimize input/output bottlenecks, and (3) consider computation bound problems only. These factors (1) through (3) restrict the PWP machine to computational problems only.

Even though the machine is designed with the conjugate gradient in mind, it will become apparent how to execute other iterative algorithms as well. The primary measure of the PWP performance is the overall time of executing the conjugate gradient for inverting eddy current data. In addition, a performance comparison between some of the current machines and the expected performance of the PWP machine will be presented. Other factors [E1] include flexibility, cost, availability and reliability will be addressed as well.

Many parallel machines utilize the memory as a common resource, the PWP machine does not rely on shared memory. Difficulties arising from shared memory are reduced at the expense of increasing the local memory. Some of these problems are synchronization, memory contention and I/O bottlenecks. By having a parallel system with sufficient local memory, the tasks can be executed locally and conflicts among the various processors is reduced. For very large problems (i.e. those problems that exceed the local memory), the PWP architecture permits memory sharing among the various processors. Hence, for a given problem of the conjugate gradient order, and with the current DSP chips, the PWP machine performs at a high computational rate. The multiple bus architecture proposed speeds the data transfer between the various units to reduce the idle time of the processors.

The machine is a linear array of processors connected by multiple bus lines. The processors which form the computational part of the machine are identical leading to a *Homogeneous system*. The main unit interfaces with the user in order to specify task sequencing. Figure 9 shows the overall architecture.

A good portion of this report is devoted to scheduling the conjugate gradient on the PWP machine. The programming is done by the user at the main unit and must be familiar with the PWP architecture to schedule the various processors. Some of the scheduling techniques available will be discussed as well as the architecture.

Figure 10 shows a single computational element (CE). The local memory of each CE is sufficiently large to execute a task without the need to request data during task execution. The CE transfers data via the multiple bus lines through the Bus Interface Unit (BIU). The number of data lines depends on the number of transfers desired and other factors such as speed of transfer and bandwidth. For the time being, the PWP architecture allows the number of bus lines to be a parameter to be chosen in later stages of the design.

Each Bus Interface Unit (BIU) has sufficient hardware to relieve the DSP from the bus control. With the local control of the BIU, the multiple busses appear as a single bus to the DSP and memory. In the next section the data transfer and processors interconnection will be presented.

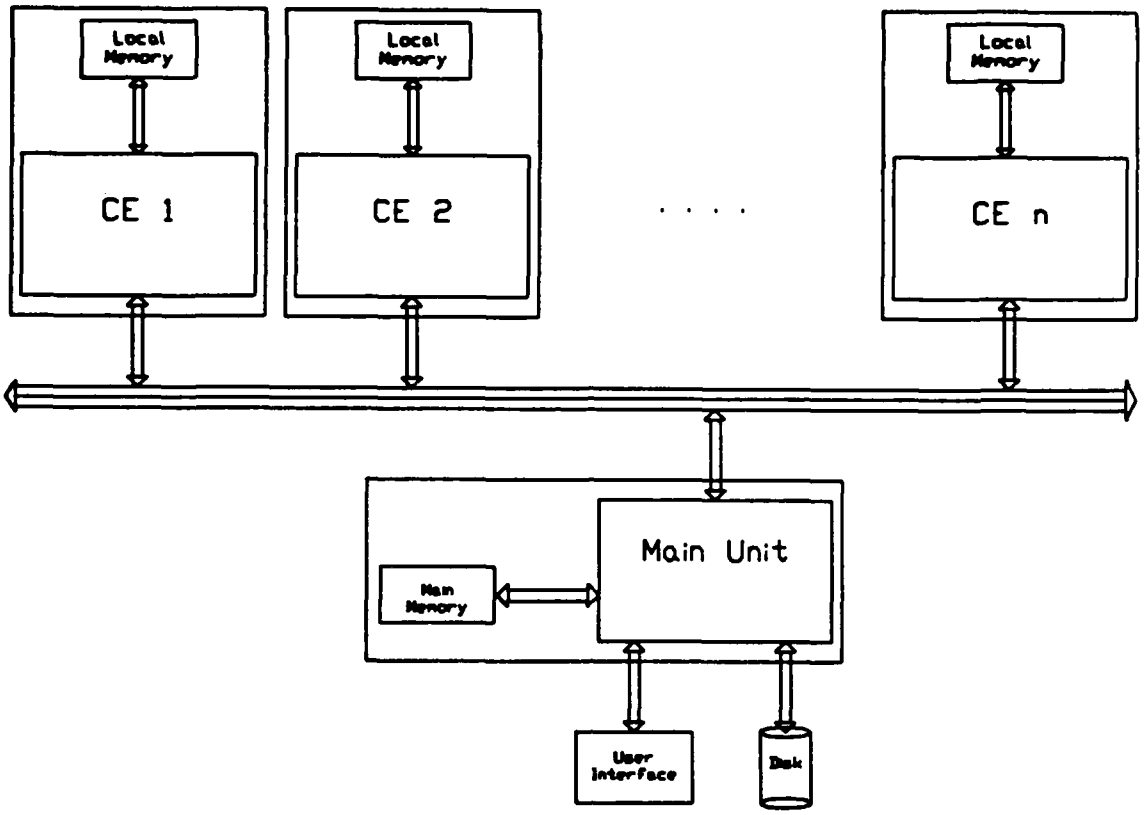


Figure 9: Overall architecture of the PWP machine. A DSP chip forms the integral computational component of each element and the computational elements are connected by multiple busses.

5. Interconnection Architecture

In this section the interconnection scheme among the computational elements is presented. The components (CE's) send I/O requests to the main unit which maintains a message queue and the bus control. The actual data transfer takes place on any of the available busses under the main unit control.

The Control Comm (CC) bus (see Figure 10) is used by all processors including the main unit. Since the main unit is the bus arbiter, additional simple circuitry is added to determine which bus is available, or if all the busses are busy. To each bus (A, B, C, and D of Figure 10), a signal indicating its activity is assigned and is denoted by BUSY. If BUSY is asserted then the bus is not available otherwise the bus is available for data transfer. At this time, the BUSY signal could be asserted only by the main unit. With multiple busses, each bus would have a BUSY line.

Figure 12 shows a simple circuit composed of logic gates to maintain the activity status of the multiple busses. The circuit schedules the busses in order so that the main

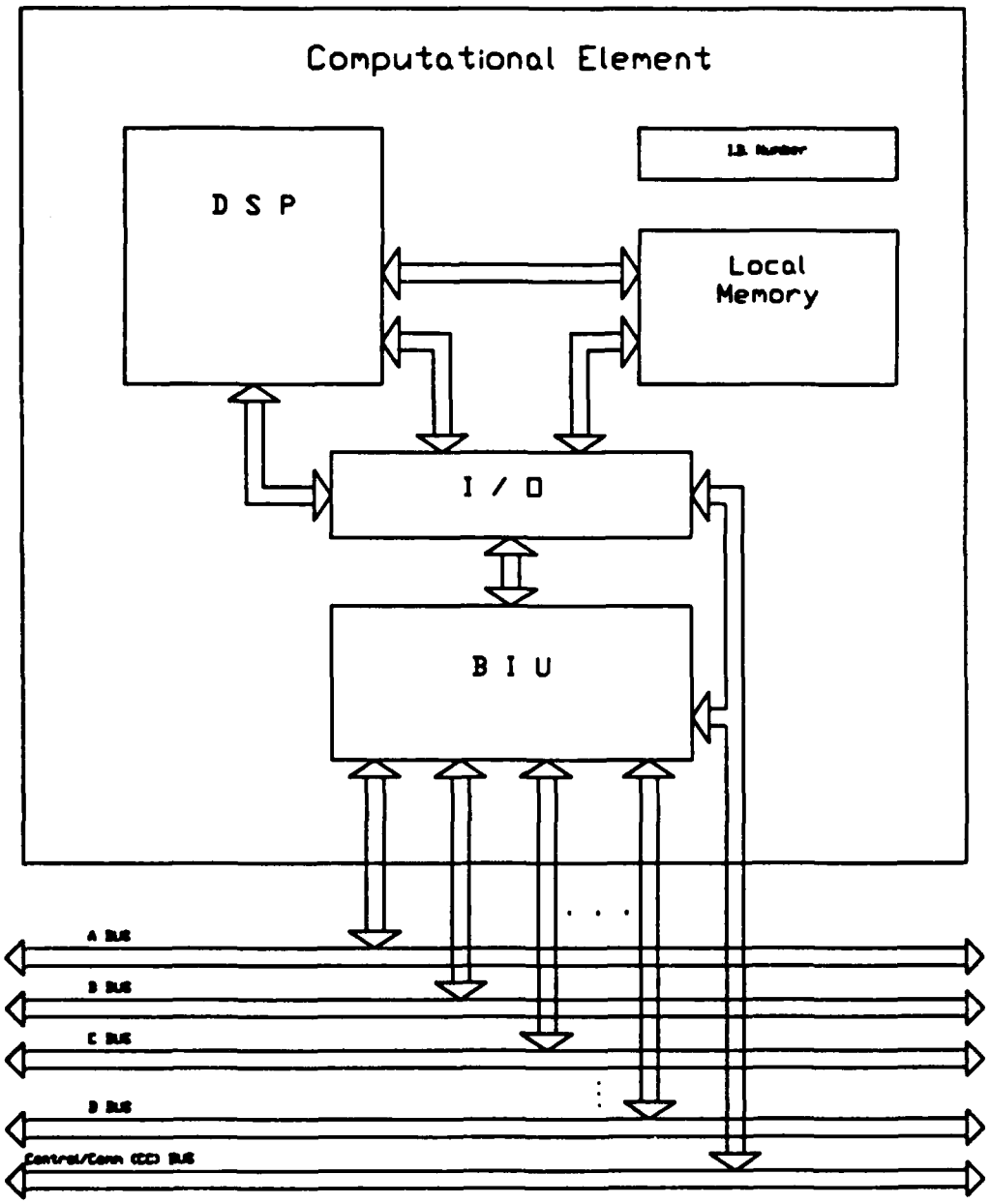


Figure 10: A computational element (CE) of the PWP machine. The data is transferred via multiple data busses controlled by the BIU. The multiple busses appear as a single bus to the components of the CE because of the local control of the BIU.

unit knows of the available busses. Only four busses are shown in Figure 12 since the circuit can be generalized in an obvious manner.

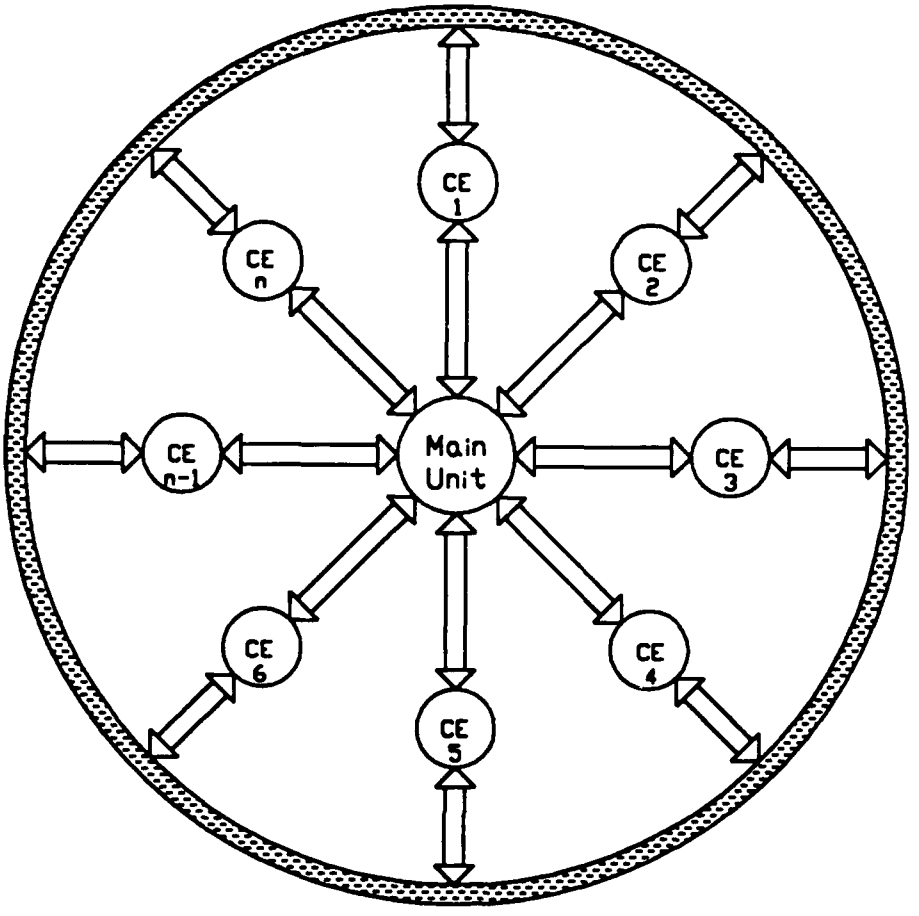


Figure 11: The communication requests are sent to the main unit to send or receive data among the components of the PWP machine. The actual data is transferred on the multiple busses (shaded area).

Denote the activity signal of the busses by BUSY-A, BUSY-B, BUSY-C and BUSY-D corresponding to the four busses shown in Figure 10. The signals EN-A, EN-B, EN-C and EN-D are the enable signals for the A, B, C and D busses respectively. The logic function associated with this circuit is to: enable Bus A for data transfer only if A is not busy, enable Bus B only if A is busy and B is not busy, enable Bus C only if A and B are busy and C is not busy, enable Bus D only if A, B and C are busy and D is not busy, and finally if all busses are busy the WAIT signal is asserted.

The enable signals switch the data bus of the CE's to the multiple busses. For example, if EN-A is asserted for a given CE, then the CE transfers data on bus A. The remaining busses transfer data in a similar manner.

The communication protocol to transfer the data among the components of the PWP machine is based on handshaking. The source requests an I/O transfer via the main unit,

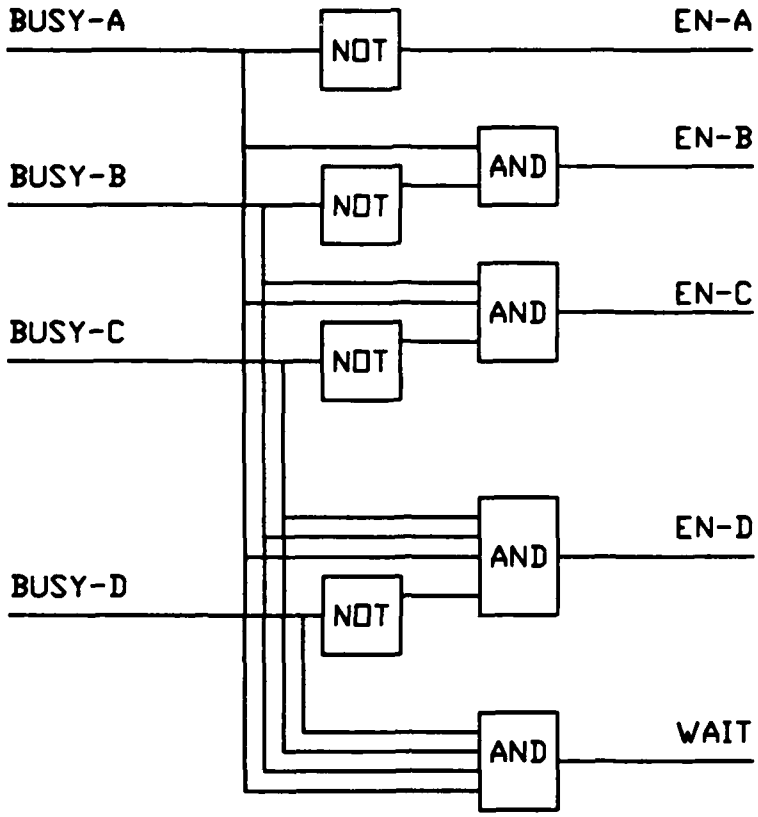


Figure 12: A simple circuit to select the next available bus. The WAIT signal is asserted only if all busses are busy.

the main unit acknowledges the request and transfers the request to the receiving CE. The receiving CE in turn acknowledges the main unit request, the main unit then holds a bus and informs both CEs of the bus. At this state, both CEs terminate the transfer via their handshake signals to inform the main unit that the transfer is terminated. The main unit then releases the bus and continues other tasks.

Since only handshake signals and requests are processed in the main unit, other I/O requests wait a minimal amount of time in order to be acknowledged. Most of the time of I/O data transfer is spent on the transfer and not the protocol. Thus the transfer of larger data with one request has only one protocol delay in comparison to smaller transfers. This justifies in part the use of sufficiently large local memory in each CE.

6. Task Scheduling

The PWP machine will accomplish the computational task for inversion of eddy currents and other computations only if the problem has been properly divided into tasks and CE sequencing. Recall that sequential problems impose a natural limit on the degree of parallelism and hence speed of execution. Task scheduling is one way of organizing

the inversion to achieve high speed and thus good performance.

Some concepts have to be explained here in order to proceed with the design. The reference is a paper by Gonzalez [G1] which will be utilized.

Measures for evaluating a parallel system have been defined in [G1] and [A1]. These measures allow us to evaluate the performance of the PWP machine. The following are the measures: (1) *completion time* of a schedule is the total time it takes to complete the schedule. A *schedule* is a collection of tasks that specify a given algorithm. For example, to invert the data, the overall inversion problem is composed of tasks in a given order that specify a schedule, (2) *flow time of a task* is the time it takes a task to complete and the (3) *flow time of a schedule* is the sum of all task times in the schedule, (4) *mean flow time* is the mean time of a task and is obtained by dividing the flow time of the schedule by the total number of tasks in the schedule, (5) *processor utilization* is equals the time a given processor is active, (6) *processor idle time* is the time a given processor is idle and it also characterizes the processor utilization, and (7) *Throughput* is defined as the number of tasks processed per unit time and is then inversely proportional to the sum of processing times of individual task sets. The measures (1) through (7) defined are used to formulate design criteria as follows [C1]:

(1) minimize finishing or completion time,
(2) minimize the number of required processors,
(3) minimize the mean flow time,
(4) maximize processor utilization, and
(5) minimize processor idle time.

While the measure in (1) is used in this report, measures (2) through (5) are less adequate for the inversion task even though they may be used to design the machine and task scheduling accordingly. The cost of the machine increases as the number of processors increases. While the cost is important, the completion time should be the dominant factor up to a certain point. As it will turn out, with the DSP implementation, the cost is reasonable when evaluated relative to the expected completion time.

Measures (3) through (5) are directly related to the programming of the parallel machine. The more these measures are weighed, the harder the programmability of the machine will be.

The following useful diagram will illustrate the various measures that depends on task times. Figure 13 shows the time chart of a parallel machine with four processors. This diagram is known as *Gantt Diagram* and it is very useful for parallel machines analysis since it could be thought of as a state diagram for the machine as time increases.

Referring to Figure 13, the time to execute a given task is enclosed in open brackets. The schedule is composed of five tasks, $T1$, $T2$, $T3$, $T4$ and $T5$. The completion time is 20 units since the schedule is completed when the second processor has completed task 5. The flow times of the tasks $T1$, $T2$, $T3$, $T4$ and $T5$ are 17, 16, 13, 3 and 8 respectively. The flow time of the schedule in Figure 13 is 57, while the mean flow time is 57/5. The

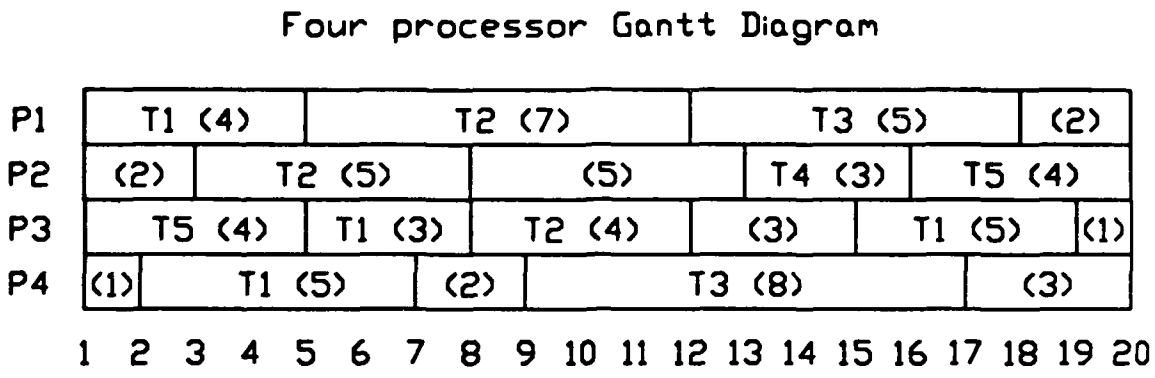


Figure 13: An example of a Gantt diagram with four processors. The diagram shows the status of the machine while executing the various tasks.

processor utilization is 18, 13, 16 and 17 corresponding to $P1$, $P2$, $P3$ and $P4$ respectively with idle times of 2, 7, 4 and 3.

While performance measures defined above characterize the machine, it can be seen that each measure reflect an aspect of the machine performance. Since the machine is for computation use and is motivated by the conjugate gradient for inversion, homogeneous or identical CE's will simplify many steps to follow. For example, if one CE performs a given task very quickly, say addition, then most of the additions will have to be performed on the given CE. This introduces difficulty in routing the addition tasks to that CE and may slow down the machine for a general computation algorithm. Thus when considering a complex task as eddy current inversion leaving the system homogeneous would simplify task scheduling. No attempt is made to survey all the work done in various areas of task scheduling, only the ones relating to the PWP architecture and constraints will be examined, inparticular [G1] and [H1].

As mentioned earlier, the completion time is the measure of performance that will be used given a fixed number of processors. Many parallel scheduling techniques assume the existence of sufficient number of processors to arrive at an optimal solution for a given problem. For example, in matrix multiplication, the number of processors depends on the size of the matrix [K3]. In matrix multiply, a processor is a multiplier or an adder. This dependency while adequate for VLSI systems, cannot be assumed for the PWP machine. As we will show later, the performance of the conjugate gradient inversion depends on the number of computing elements. In particular, the number of frequencies and layers determine the number of computing elements. However, once the number of computing elements is determined for an inversion problem, it cannot change, and the computing elements have to be scheduled to solve any inversion problem. Since the computing elements are general computational devices with their supporting memory and hardware, specifying their number should be approached with care. The number of CEs should be determined by the size of the most frequent inversion problem in order to decrease hardware cost.

There are two basic scheduling strategies: (1) *Preemptive* and (2) *Non-Preemtive* or *Basic*. Preemptive scheduling deals with creating schedules that admit task interruption and manages task priorities. Basic Schedules are the schedules that do not admit task interruption. The conjugate gradient, or most computation type problems, seem to fall under basic scheduling because computation bound problems (specifically conjugate gradient) can be scheduled according to a specific algorithm with known execution time. The basic idea of the PWP machine is to solve computation bound problems and not handle tasks that deal with interrupts such as multi user or external interrupt. Thus only basic schedules will be considered.

The following scheduling algorithm is adopted from [H1] and has been adapted to work with the conjugate gradient. The next section will consider both the constrained and un-constrained conjugate gradient inversion schemes. Now, a general terminology and methodology is presented [H1].

Given n tasks representing the inversion algorithm with a partial order. It is desired to know how to arrange the tasks, given a fixed number of processors, and given a set of tasks, how to find the minimum number of processors to execute them. In [H1] it was assumed that all the tasks take an equal amount of time to complete and no cycles are present in the graph. In what follows, these assumptions will be relaxed to include tasks of unequal duration and cycles. First Hu's paper will be explained in short detail to provide the terminology and the scheduling solution under the assumptions stated.

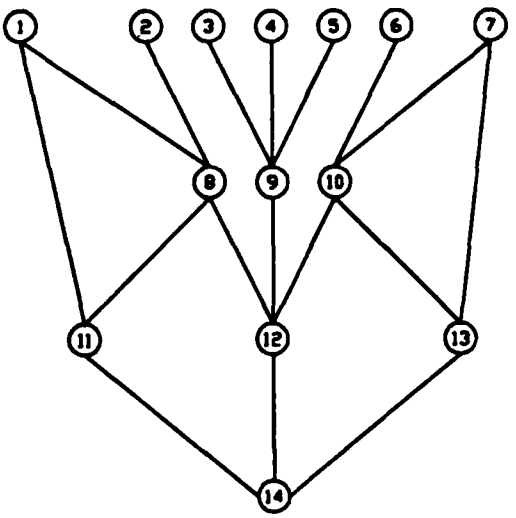


Figure 14: An example of an algorithm represented by a graph. Note that the graph contains no cycles.

Consider an algorithm represented by a graph as in Figure 14 with no cycles and each task time is one unit. A partial order, denoted by $<$, is defined by the directed arcs. For example, $N_2 < N_8$, $N_1 < N_8$ meaning that task 2 and task 1 have to be completed before task 8 begins. Note that $N_8 < N_{12}$ and $N_8 < N_{11}$ but N_{11} is not related to N_{12} and that

task 11 and 12 can be executed independently of each other.

Hu's sequencing algorithm is to assign labels to the nodes of the graph according to the following rules:

(1) A node N_i is labeled with $\alpha_i = l_i + 1$ if l_i is the length of the longest path from node N_i to the final node G. The final node G receives the label of 1 since the length to itself is zero. If there is more than one final node, an empty node E is created so that all final nodes precede E. All adjacent nodes receive a label of 2. If the longest path from node N_i to the final node G is not unique, then the choice is arbitrary and any one longest path is admitted.

(2) Sort the nodes according to the labels given in step (1) and redraw the graph with all nodes of the same label are on the same level.

(3) Choose any m nodes of the graph such that these nodes are not the predecessor of any other nodes and m is the number of available processors.

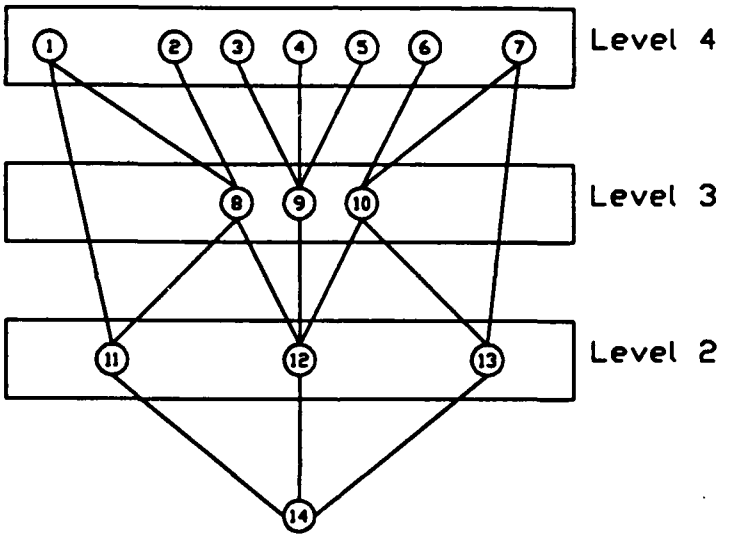


Figure 15: Graph of Figure 14 after labeling according to steps (1) and (2) of Hu's sequencing.

When steps (1) pand (2) are performed on the graph in Figure 14, the resulting graph is shown in Figure 15. Note the partial ordering is still preserved, and step (3) identifies the tasks with the available processors. Suppose there are m processors available, so the m tasks are removed from the graph in order to be completed. After one time unit, the graph reduces to Figure 16 when m is equal to four. This example will be carried further to illustrate some concepts that will arise for the conjugate gradient. The Gantt diagrams (in Figure 17) corresponding to Hu's sequencing of the graph shows the completion time as the number of processors increases from one to seven. The completion time remains the same (five time units) for the case of four through six processors. The

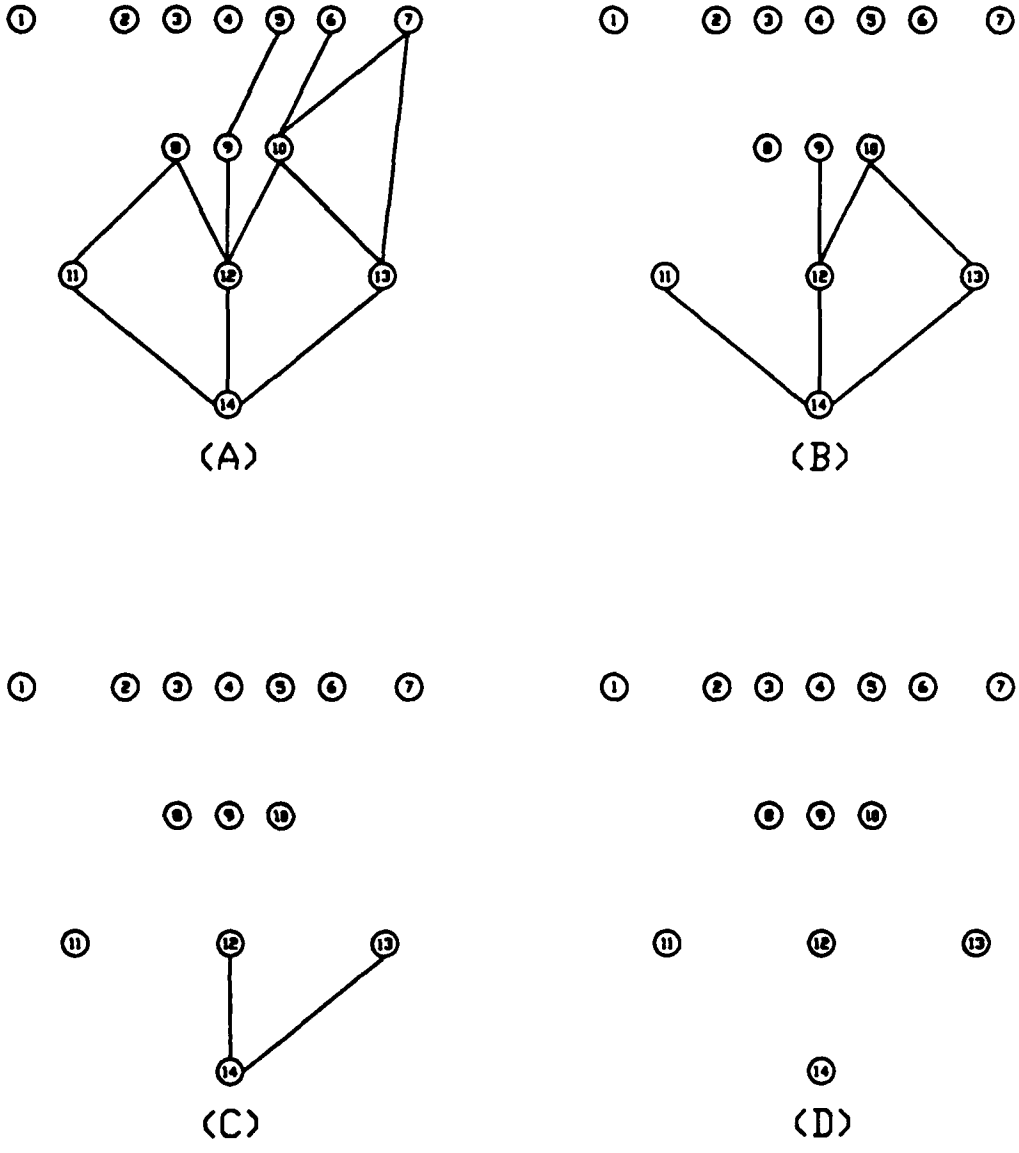


Figure 16: Hu's scheduling of the graph in Figure 14. The number of available processors is equal to four, hence only 4 nodes are removed at a time to yield (A) through (D).

minimum completion time is obtained when seven processors are used, which correspond to the largest number of nodes over all levels obtained from Hu's sequencing.

In relation to the pipeline architecture, if the number of computation layers is to equal to the number of levels, then we would require a larger number of processors to use the pipeline architecture for minimum completion time. For the example, we would require seven, three, three, and one processor (total of fourteen) corresponding to Layer 1 through Layer 4 respectively. With this configuration, the throughput would be one

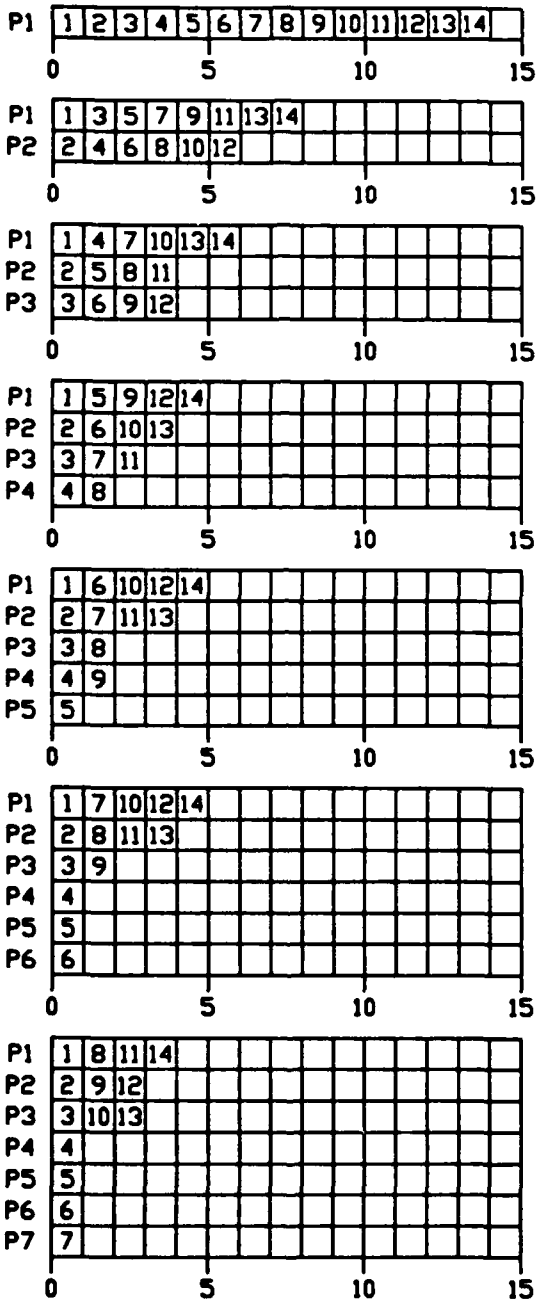


Figure 17: Gantt diagrams of the example when executed on a parallel machine. As can be seen, increasing the number of processors may not always decrease the completion time.

schedule per unit task time. The completion time measure is plotted in Figure 18 versus the number of computing elements. The graph is valid only for the example, but it illustrates the point that a parallel machine depends on the algorithm, and in particular, on the tasks partial order.

The conjugate gradient inversion will be analyzed in a similar manner in the following sections. First the cycles and the problems they introduce are discussed.

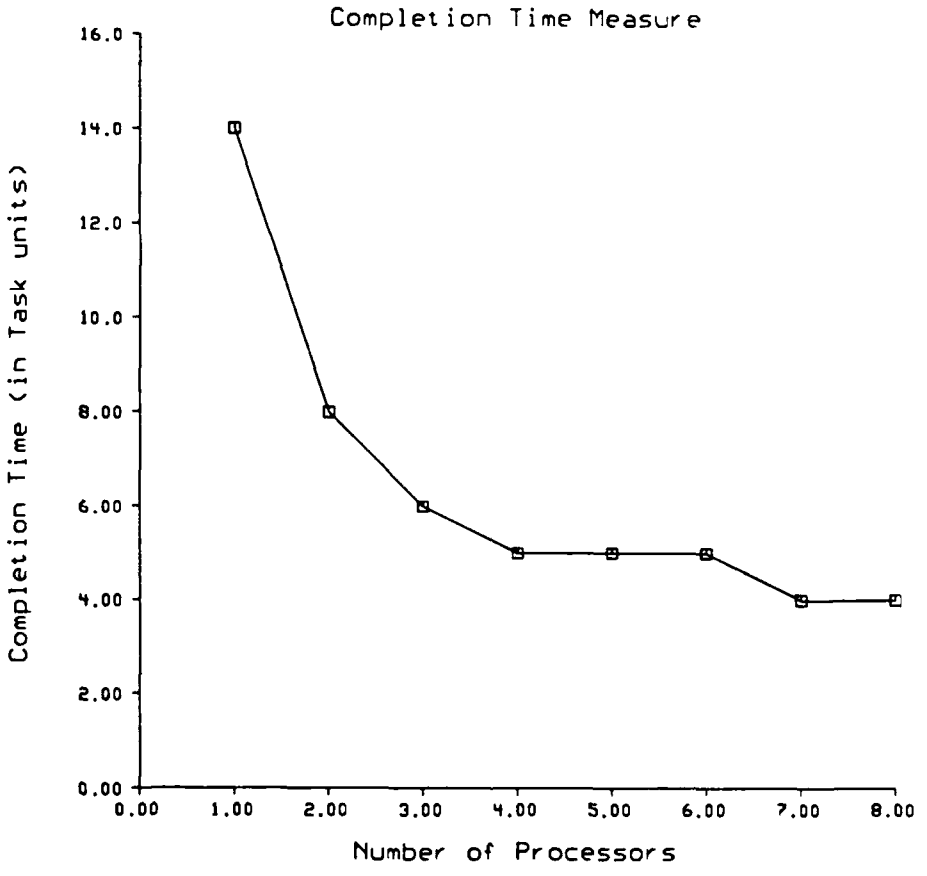


Figure 18: The completion time is not a monotone decreasing function of the number of processors. Note the flat region between four and six processors.

6.1. Cycles in Algorithm Graphs

In computational algorithms, a set of tasks is usually repeated many times. The repetition appears as cycles in the graph of an algorithm. When considering a pipeline machine, or the PWP machine in the pipeline configuration, cycles introduce problems to the pipeline architecture. The problem is the fact that if the cycle duration is large compared to the other tasks, then the other tasks will not propagate through the pipeline but

will wait for the cycle to terminate. This behavior slows down the parallel machine since the processors not executing the cycle will be idle. In the following, we shall present some methods to handle this situation and in particular the cycles present in the constrained conjugate gradient.

The first scheme which worked for the conjugate gradient is to eliminate the cycle. A detailed analysis of the modified algorithm is in Appendix A of this chapter. Cycle elimination method is very restrictive and cannot be applied to arbitrary algorithms. An alternate scheme was developed to handle cycles in more general settings.

The cycle could propagate through the pipe if every processor in the pipeline can execute the tasks in the cycle. Hence, with the PWP machine, every processor executes the cycle once and transfers the data to the next processor in the pipe. This scheme is not restricted to the PWP machine, but to all parallel machines that can execute all tasks in the schedule or at least the tasks specified by the cycle. As the cycle is propagating through the pipeline, other processors can execute tasks in the FIFO queue.

6.2. Decomposition of The Inversion Algorithm

In analogy with assembly line processing, it would be advantageous to keep the computing elements operating on a fixed task. Fixing the task in each processor eliminates the reconfiguration the processors which would reduce the completion time. An example where the reconfiguration could be eliminated is when cycles propagate through the pipe. The PWP machine is very flexible and could programmed in many configurations.

The PWP machine has been designed with the CG in mind. It is also suitable for problems that are computationally bound, and could be decomposed into tasks. If the problem cannot be decomposed, then the pipeline configuration permits the increase in throughput for more than one problem. The study and analysis of this problem can be generalized to include iterartive algorithms as well. The main idea is to take advantage of the parallelism that exists in the algorithm but not to a very low level. Low level parallelism complicates the system and increases the cost, and increased speed may not justify the additional complexity.

To specify the inversion tasks, one must consider the equations defining the conjugate gradient. There are two basic algorithms: constrained and unconstrained. Both algorithms invert the data and are different. Initially the unconstrained case is studied followed by the constrained case.

6.2.1. Unconstrained Task Decomposition

Recall the $\bigcirc$ operator was defined in Equation (9) to be the sum of convolution of matrices. This operation will be considered in this section for implementation on the PWP machine. The convolution in Equation (9) reduces to matrix product when the convolution operator is replaced by matrix product. This fact is exploited next in order to implement the $\bigcirc$ operation using algorithms that have already been developed for matrix product. One algorithm that will be considered is obtained from [K3] denoted by *Synchronized Matrix Product* algorithm. First an example with five layers ($N_z = 5$) and four frequencies ($N_f = 4$) will be considered, followed by the general case. Start with the

following equations (see Equation 9):

$$(A\ OX)_1 = T_{11} * X_1 + T_{12} * X_2 + T_{13} * X_3 + T_{14} * X_4 + T_{15} * X_5 \quad (10)$$

$$(A\ OX)_2 = T_{21} * X_1 + T_{22} * X_2 + T_{23} * X_3 + T_{24} * X_4 + T_{25} * X_5 \quad (11)$$

$$(A\ OX)_3 = T_{31} * X_1 + T_{32} * X_2 + T_{33} * X_3 + T_{34} * X_4 + T_{35} * X_5 \quad (12)$$

$$(A\ OX)_4 = T_{41} * X_1 + T_{42} * X_2 + T_{43} * X_3 + T_{44} * X_4 + T_{45} * X_5 \quad (13)$$

$$(A\ OX)_5 = T_{51} * X_1 + T_{52} * X_2 + T_{53} * X_3 + T_{54} * X_4 + T_{55} * X_5 \quad (14)$$

$$(A\ OX)_6 = T_{61} * X_1 + T_{62} * X_2 + T_{63} * X_3 + T_{64} * X_4 + T_{65} * X_5 \quad (15)$$

$$(A\ OX)_7 = T_{71} * X_1 + T_{72} * X_2 + T_{73} * X_3 + T_{74} * X_4 + T_{75} * X_5 \quad (16)$$

$$(A\ OX)_8 = T_{81} * X_1 + T_{82} * X_2 + T_{83} * X_3 + T_{84} * X_4 + T_{85} * X_5 \quad (17)$$

The number of processors affects the performance to a great extent. As the number of processors increase, the performance may or may not improve as was illustrated in the previous example and will be observed in the conjugate gradient. If the number of processors is too low, the total inversion time may not be adequate. Further more, the speed will depend on particular parameters of the inversion, for example, fixing the parameters will give a number of processors that will be optimal or suboptimal for those parameters. If any of these parameters change, then the machine may or may not have better performance.

Equations (10) through (17) are represented by the following graph which can be analyzed and scheduled.

Applying Hu's scheduling scheme with five processors, Figure 20 is the corresponding Gantt diagram. The diagram illustrates many points which we discuss now. Observe that with five processors, in the first eight time units all the processors perform the same task namely convolution. The transfer during this time involves only the data and there would be no task reconfiguration. The later seven time units perform only addition. The assumption of equal task time fails when the processors switch from convolution to addition. Since the later tasks are only addition and take equal time to complete on data of the same size, another time unit could be defined. This behavior is a characteristic of the conjugate gradient as illustrated by Figure 19. We can add a performance measure to be *the number of task switching* per processor. This measure is related implicitly to the completion time. It should be included in the evaluation process when all factors affecting the performance are considered in the scheduling process. An important second order consideration is the time it takes the data to go from a source node to a sink node and the waiting times for synchronization if any.

Now we examine the completion time as a function of the number of processors used to execute the conjugate gradient. Figure 21 is the completion time measure of the unconstrained conjugate gradient for four frequencies and five layers.

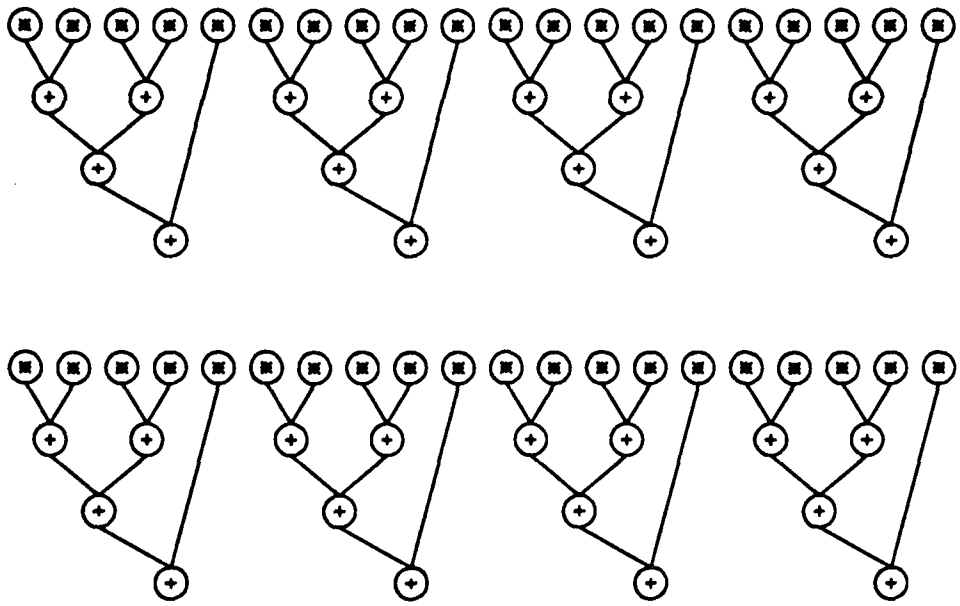


Figure 19: The algorithm graph of the unconstrained conjugate gradient inversion for four frequencies ($N_f = 4$) and five layers ($N_z = 5$).

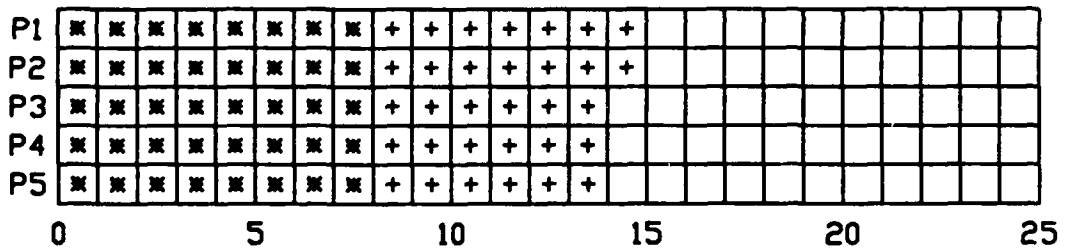


Figure 20: Gantt diagram for unconstrained conjugate gradient executed with five processors ($N_f = 4$, $N_z = 5$).

Figure 21 shows that the most improvement occurs around 10 processors for that particular case. The slow decay in the completion time curve is attributed to the inherent dependencies of the conjugate gradient. To obtain the actual expected inversion time, the completion time is multiplied by the total number of iterations. The number of iterations is a function of the sampling frequency in the spatial domain, that is, it depends on the number of points in the measurement space. Our experiments with the conjugate gradient, suggest that convergence occurs around 2000 iterations as illustrated in an earlier chapter. We can estimate, based on the DSP chips, that each convolution task will take approximately 3 milliseconds, to give an estimate of 36 seconds for 2000 iterations.

The algorithm for the general case is similar to the graph in Figure 19. There would be $2N_f$ trees each with N_z leaves to give a total of $2N_fN_z$ terminal nodes (or leaves).

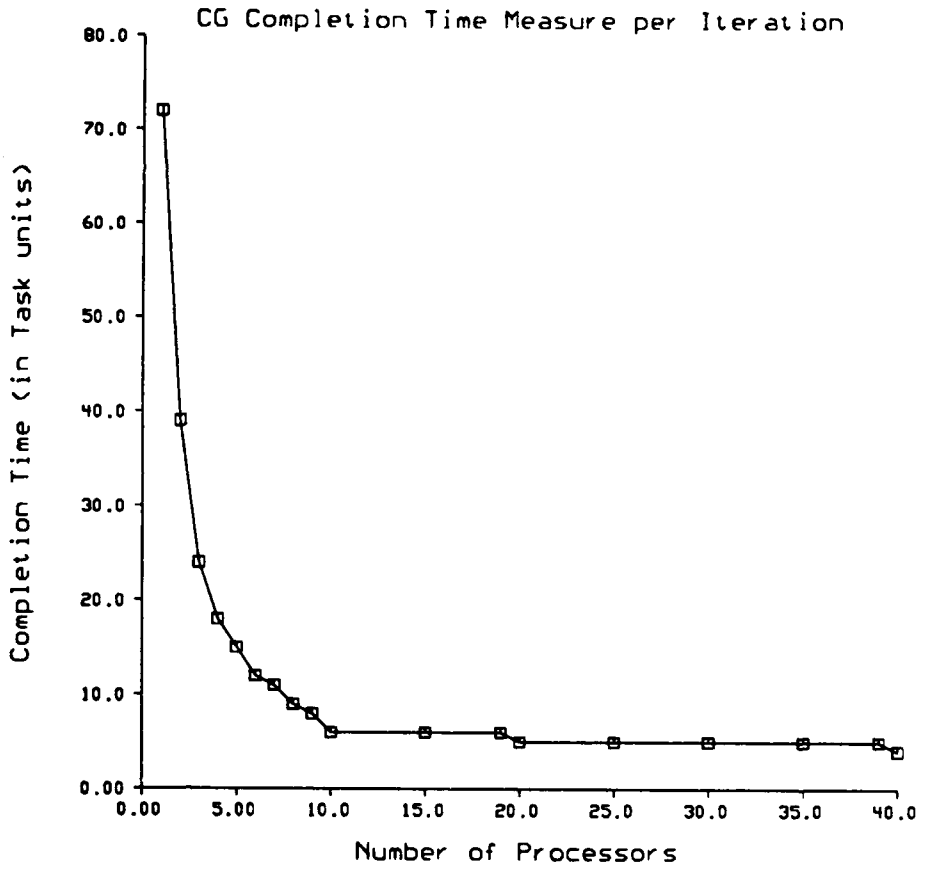


Figure 21: Completion time of the convolution step of Equations (10) through (19) for four frequencies and five layers.

Applying Hu's algorithm to the general conjugate gradient, yields, that if the number of processors equal to $2N_fN_z$, then the completion time is four task times. This number of processors is large and after the first task, almost half of the processors are idle and it rises sharply afterward. A conservative system would be to use $2N_z$ processors which attain a completion time of six time units.

The actual times for evaluating Equation (9) defining $\bigcirc$ were compiled for the conjugate gradient running on the Alliant FX/1. The Alliant times are the actual times it took to evaluate both $A \bigcirc X$ and $A^* \bigcirc X$. The completion time of performing the same computations on the PWP machine are listed in Table 4 on the following page and compared with the Alliant. The PWP performance was estimated when the number of CEs is eight and Twenty.

Each CE is assumed to take 2 and 1 milliseconds for matrix convolution and matrix addition respectively. The Alliant figures include the overhead increase due to its multi-

Table 4: Calculation times of $A \bigcirc X$ and $A^* \bigcirc X$ on the Alliant FX/1 Computer compared with the expected time on the PWP machine.

Inversion parameters			Alliant FX/1	PWP			
				Eight CEs		Twenty CEs	
N_f	N_z	Iterations	Time (sec)	Time (sec)	Ratio	Time (sec)	Ratio
5	4	100	932	2.8	332	0.8	1165
5	4	100	892	2.8	318	0.8	1115
10	4	100	1519	5.6	271	2.2	690
10	4	100	1509	5.6	269	2.2	685
5	4	2048	18302	57	321	16	1143
9	4	2048	29222	114	256	45	649

user environment. So the actual time could be slightly less. The PWP times did not include the I/O transfer time, scheduling time, or other overhead. Thus the PWP times should increase slightly. The improvement ratio would be around 1100 and 650 if these slight variations are included.

7. Simulation of Computer Systems

As mentioned earlier, the performance depends on both the software and the hardware of the computer. In this section, we describe a simulation tool of both aspects. Petri nets, a mathematical model of concurrent systems, have gained increased usage and acceptance since their introduction in early 1960's, as a tool for modeling asynchronous concurrent systems [P1]. They have been used to model program structures in [A1] and hardware in [P1] as well as communication protocols in [M1]. Torn [T1] has extended the notion of Petri nets to *Simulation Graphs* and used *SIMULATION*, a process-oriented language, to code the graphs obtained from Petri nets. These are the components for modeling and simulating the PWP machine.

A *Petri Net* is a mathematical model of systems. As with any model, the analysis of the Petri net model can reveal important information about the system and may suggest improvements as the model parameters are varied (for example, changing the bus lines of the PWP machine, or task sequencing). A brief overview of Petri nets is presented following the excellent papers of Peterson [P1] and Torn [T1].

7.1. Overview of Petri Nets

A *Petri Net* is a four tuple structure (P, T, I, O), where P is a set of *places*, T is a set of *transitions*, I is an *input* function, and O is an *output* function. The set of places P and the set of transitions are both non-empty, finite and disjoint. For each transition $t_i \in T$, the input function I, defines the set of *input places* p_i and the output function O, defines the set of *output places* p_i.

A graphical representation of a Petri net is more useful for observing the behavior of modeled systems. To each Petri net, a *Petri Net Graph* can be constructed. The graph is composed of circles O and bars | representing places and transitions respectively. The input and output functions are represented by directed arcs from transitions (bars) to places (circles) and from places to transitions respectively. For example, the following is a Petri net of a single-queue single-processor system and its corresponding graph.

$$C = (P, T, I, O)$$

$$P = \{p_1, p_2, p_3, p_4\} \qquad T = \{t_1, t_2, t_2, t_4\}$$
$$I(t_1) = \{\ \} \qquad O(t_1) = \{p_1\}$$
$$I(t_2) = \{p_1, p_2\} \qquad O(t_2) = \{p_3\}$$
$$I(t_3) = \{p_3\} \qquad O(t_3) = \{p_2, p_4\}$$
$$I(t_4) = \{p_4\} \qquad O(t_4) = \{\ \}$$

Figure 22. Petri net structure of a single-queue single-processor computer system.

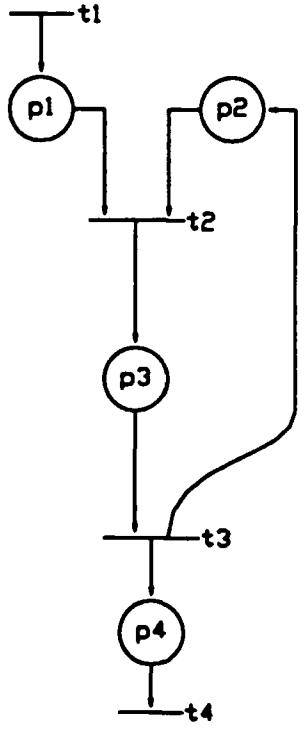


Figure 23. Petri net graph representation corresponding to the structure in Figure 22 above.

The example illustrated in Figure 5 is useful in modeling and simulating a queuing system. The events could represent the following sequence of operations [P1]: (1) Tasks enter the queue, (2) tasks start to execute, (3) tasks complete execution, and (4) tasks leave

the queue. Note that events (1) through (4) are represented by transitions t_1, t_2, t_3 and t_4 respectively. The places of interest are: (a) A task on the list, (b) an idle processor, (c) a task being processed, and (d) a task leaves the queue. These places are represented by p_1, p_2, p_3 and p_4.

An important feature of Petri nets is their hierarchical aspect of modeling a system. The hierarchical nature of Petri nets permits modeling from coarse to fine levels depending on the desired detail. For example, concurrent processes can be expressed in terms of the single-server (main unit) multiple-processor queue as shown in Figure 22.

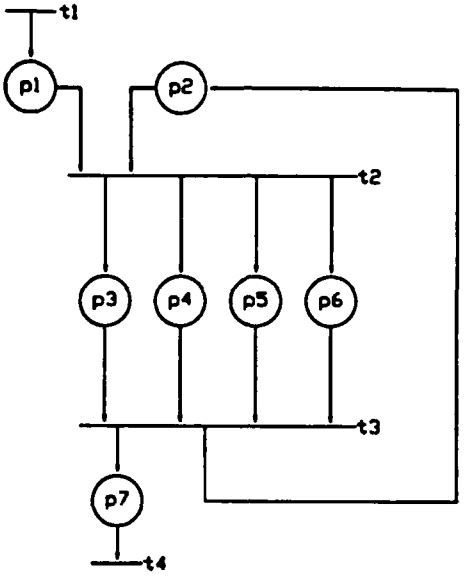


Figure 24. Hierarchical property of a Petri net used to model concurrent tasks.

Finally, we introduce two concepts [P1] of Petri nets to derive a model net for the multiple bus lines utilized by the PWP machine. The first is that of a *Token* which is an element of a Petri net assigned to places and is represented by • on the corresponding graph. Tokens are passed among the places when a given transition is fired. The firing of a transition marks an event which causes the tokens to move from the input places to the output places. The execution of a Petri net is determined by the number and distribution of tokens in the net.

The second concept of a Petri net is that of an *inhibitor arc*. An inhibitor arc from a place p_i to a transition t_j enables the transition t_j if there are no tokens in the place p_i. Inhibitor arcs are denoted by a small circle O in place of the arrow head of the input or output functions, and they extend the modeling power of a Petri net to include priorities and mutual exclusion of events. Figure 25 illustrates token passing, transition firing, inhibitor arcs and it is assumed that the net is part of a larger system.

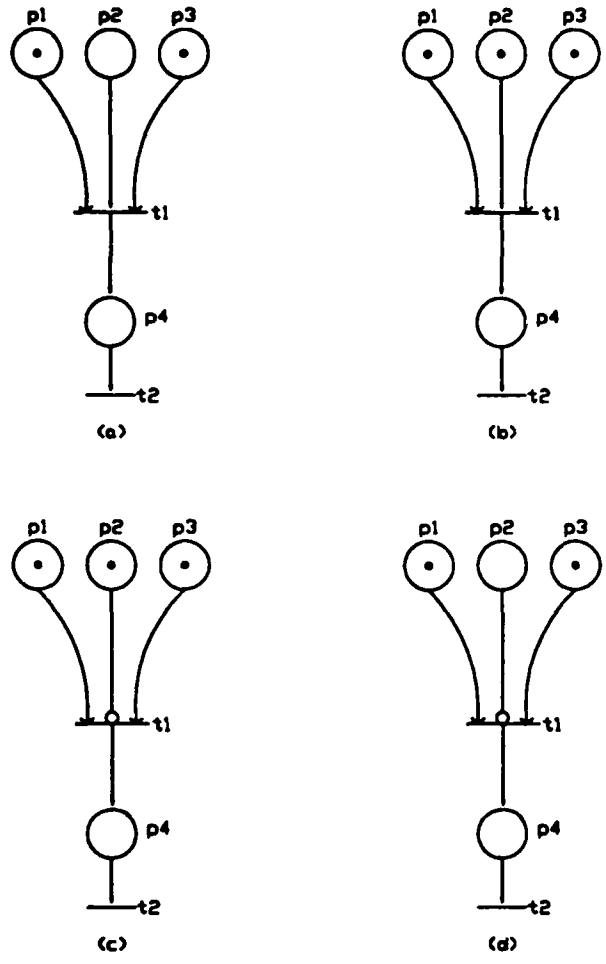


Figure 25. (a) transition t_1 cannot fire since there is no token in place p_2. (b) transition t_1 fires when there are tokens in all it's input places, and which in turn moves the tokens into place p_4. (c) transition t_1 cannot fire since there is token and an inhibitor arc in place p_2. (d) transition t_1 fires since there is no token in place p_2 and each of the places p_1 and p_2 has a token.

As mentioned earlier, a Petri net is derived to model the data transfer between the processors. The main unit processes the I/O request and the communication protocol between any computing elements. The actual transfer occurs along any of the available data busses. Figure 8 illustrates the I/O management done by the main unit on three bus lines (A, B, and C of Figure 3). Transition t_4 has higher priority than t_3 and t_6 as dictated by the inhibitor arcs shown. The structure of the net is that of a single server (main unit) multiple processors (the bus lines) with priority. The places p_4, p_5 and p_6 are the three bus lines that transfer the data when enabled. The model shows that places (where the transfer occurs) p_4, p_5 and p_6 will be occupied in priority as well. The overall effect is that, bus A will be used if it is available (p_4), bus B (p_5) will be used only if bus A is busy, and bus C (p_6) will be used only if bus A and B are busy. If all three busses are

busy, then the transfer requests are placed in a queue at the main unit.

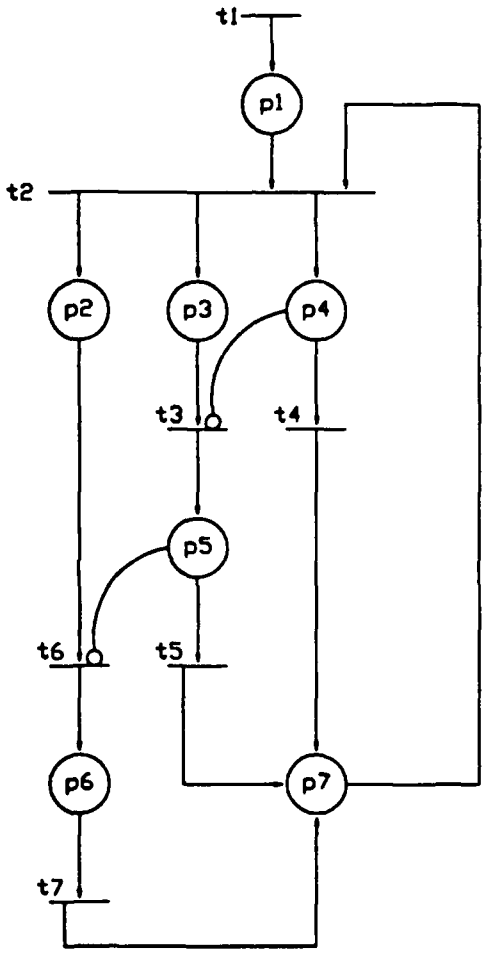


Figure 26. Petri net model of the multiple bus architecture (three busses here).

8. Summary of Chapter VII

In this chapter, a parallel machine to solve iterartive problems has been designed. In particular, the conjugate gradient inversion algorithms.

The first sections discussed the size of the inversion problem. An estimate of the memory requirement was obtained in order to evaluate the feasibility of the PWP machine. The requirement depended on the inversion problem and it varied linearly in each of the inversion parameters to give an overall requirement of the fourth order. Even with this requirement, the size was reasonable for typical problems encountered in NDE. If the problem of interest is large relative to the available memory, then the problem must be partitioned on the PWP machine. This particular problem is inherent in all computers when the memory requirement is larger than the memory resources.

An architecture for pipeline configuration was discussed. We also discussed how to improve the convergence using the pipeline architecture. The pipeline was composed of computational layers with several computing elements in each. The task sequencing scheme can be used to determine the number of computing elements for each layer in the pipe.

A parallel machine utilizing DSP chips was designed and with the multiple bus architecture, the I/O problems can be reduced. The overall speed of the system depends on the speed of the individual DSP chips used to integrate the system. Furthermore, as more advanced DSP chips become available, the system could be upgraded with minimal changes.

In addition, we considered a scheduling scheme to assign tasks in order to execute the conjugate gradient. The sequencing scheme was applied to our inversion algorithm, and we obtained estimates of the completion times. The completion time per iteration was six time units for $2N_z$ computing elements, five for $N_f N_z$ processors and four units for $2N_f N_z$. The cost of the computing elements becomes very high when the desired completion time is four units as illustrated in Figure 21. When cost, scheduling and speed are considered, $2N_z$ processors would be the trade off number of processors to use.

Having chosen a number of processors for a particular inversion problem, the scheduling scheme discussed minimizes the completion time of an inversion problem given a fixed number of processors. The completion time for the new problem depends on its parameters. In Table 4 on page 30, we compared the expected performance of the PWP machine to the actual Alliant FX/1 performance. A speed up factor on the order of 1100 is noted for the considered cases.

Finally, a simualtion tool for concurrent systems was presented. By simulating the PWP machine architecture, we can evaluate its performance and improve the design. The goal of the simulation is to minimize the completion time and cost.

BIBLIOGRAPHY

[A1] Ashour, S., *Sequencing Theory*, Lecture Notes in Economics and Mathematical Systems, No. 69, edited by M. Beekmann, Providence, G. Goos, Karlsruhe, and H. P. Kunzi, Springer-Verlag, 1972.

[A2] T. Agerwala, *Putting Petri Nets to Work*, Computer, IEEE, December 1979.

[C1] Conway, R. W., Maxwell, W. L. and Miller, L. W., *Theory of Scheduling*, Addison-Wesley Publishing Co. Inc., Reading Massachusets, 1967.

[E1] Enslow Jr., Philips, H., *Multiprocessor Organization - A Survey*, ACM Computing Surveys, Vol. 9, No. 1, March 1977, pp 103-129.

[G1] Gonzalez Jr., Mario, J., *Deterministic Processor Scheduling*, ACM Computing Surveys, Vol. 9, No. 3, September 1977, pp 173-204.

[H1] Hwang, Kai and Briggs, Fae, *Computer Architecture and Parallel Processing*, McGraw-Hill, 1984.

[H2] Hu, T. C., *Parallel Sequence and Assembly Line Problems*, Operation Research 1961, pp 841-848.

[K1] Kawata, S. and Nalcioglu, O., *Constrained Iterative Reconstruction by the Conjugate Gradient Method*, IEEE Transactions on Medical Imaging, Vol MI-4, No. 2, June 1985, pp 65-71.

[K2] Kung, H. T., *Why Systolic Architectures?*, Computer, January 1982, pp 37-46.

[K3] Kronsjo, Lydia, *Computational Complexity of Sequential and Parallel Algorithms*, John Wiley & Sons Publishing Co., 1985.

[M1] Mano, M. Morris, *Computer System Architecture*, second edition, Prentice-Hall, 1982.

[M2] P. M. Merlin, *A Methodology for the Design and Implementation of Communication Protocols*, IEEE Trans. on Comm., Vol. COM-24, No. 6, June 1976.

[N1] G. J. Nutt, *Evaluation Nets for Computer System Performance Analysis*, Fall Joint Computer Conference, 1972.

[P1] Peterson, James L., *Petri Nets*, ACM Computing Surveys, Vol. 9, No. 3, September 1977, pp 223-252.

[S1] Seager, Mark K., *Parallelizing conjugate gradient for the CRAY X-MP*, Parallel Computing, Vol 3, 1986, pp 35-47.

[T1] A. A. Torn, *Simulation Graphs: A General Tool for Modeling Simulation Designs*, Simulation, Vol. 37, No. 6, December 1981.

[V1] Van der Vorst, Henk *The performance of FORTRAN implementations for preconditioned conjugate gradients on vector computers*, Parallel Computing, Vol 3, 1986, pp 49-58.

Appendix A

The Conjugate Gradient Algorithm for a Pipelined Architecture

by Fouad Mrad

The original algorithm representing the conjugate gradient minimization procedure is not preferred for a pipelined architecture. The reason for that is caused by the internal looping around one of the intermediate steps. Since that problem affects greatly the efficiency of the architecture, we had to find a way to deal with this obstacle. Many solutions had to be examined, from software manipulations to changing the structure of the whole algorithm. A new version of the conjugate gradient algorithm for constrained problems is presented, the correctness of this version and its improved efficiency were supported by theoretical, as well as, numerical means.

The following function is to be minimized

$$F(X) = \frac{1}{2} \| Y - A \bigcirc X \|^2.$$

Start with an initialization of an iteration counter k, $k = 1$.

Step1. Select a point X_1 in S. Compute

$$R_1 = Y - A \bigcirc X_1,\ Q_1 = A^* \bigcirc R_1 = -F'(X_1),\ g_{i1} = g_i(X_1)\ (i = 1, \ldots, 2N). \quad \text{(a)}$$

If $Q_1 = 0$, **stop** ; algorithm terminates.
Else, let I be the set of indices, i, such that $g_{i1} = 0$ (the 'active set'). **Go to Step 2.**

Step2. If I is empty, i.e., no active constraints, let H be the identity matrix. **Go to Step 3.**
Else, let H be the nonnegative symmetric matrix that annihilates the vectors, W_i, $i \in I$.
If $H = 0$ **go to Step 5,** with X_1 playing the role of X_2.
Else **go to Step 3.**

Step3. CG-subroutine. Set

$$P_1 = \bar{Q}_1 = HQ_1. \quad \text{(b)}$$

$$S_1 = A \bigcirc P_1,\ a_1 = \frac{\| \bar{Q}_1 \|^2}{\| S_1 \|^2} \quad \text{(c)}$$

$$l_{i1} = \begin{cases} W_i^T P_1, & i \notin I \\ 0, & i \in I \end{cases} \quad \text{(d)}$$

$$X_2 = X_1 + a_1 P_1,\ g_{j,2} = g_{j1} + a_1 l_{j1}\ (j = 1, \ldots, 2N) \quad \text{(e)}$$

If for *some* $j \notin I$, $g_{j,2} \geq 0$, *then Set* k = 1 and **go Step 4.** (f)

Else if k = N then

$$R_2 = R_1 - a_1 S_1,\ Q_2 = H A^* \bigcirc R_2,\ \text{k} = 1 \text{ and } \textbf{go Step 5.} \quad \text{(g)}$$

Else

Replace X_1 *by* X_2, and k *by* $k + 1$ and **go Step 1.** (h)

Step 4. Scale back to the boundary of the feasible region and update the active set.
Let J be all indices $j \notin I$, *such that* $g_{j,2} \geq 0$. Let $\bar{a}_1$ be the smallest of the ratios

$$a_{j1} = -\frac{g_{j1}}{l_{j1}},\ j \in \mathrm{J}.$$

Reset

$$X_1 = X_1 + \bar{a}_1 P_1,\ R_1 = Y - AoX_1,$$

$$Q_1 = A^* \bigcirc R_1 = -F'(X_1),\ g_{i1} = g_{j,1} + \bar{a}_1 l_{j1},\ (j = 1, \ldots, 2N).$$

Update the active set by adjoining to I all indices $j \notin \mathrm{I}$, and $g_{j1} = 0$. **Go to Step 2.**

Step 5. If $Q_2 = 0$, **stop;** algorithm is terminated.
Else select shortest V of the form

$$V = Q_2 - \sum W_i \lambda_i,\ (i \in \mathrm{I}\ with\ \lambda_i \geq 0) \qquad \text{(i)}$$

If V = 0,**stop;** Kuhn-Tucker conditions are satisfied, and algorithm is terminated at minimum point of F on S.
Else, choose $a > 0$, such that

$$g_{j,2} + a\ V^T W_j \leq 0\ (j = 1, \ldots, 2N),\ F(X_2 + a\ V) < F(X_2). \qquad \text{(j)}$$

Restart the algorithm at **Step 1** with $X_1 = X_2 + a\ V$ as the initial point.

Theoretical Note : Comparing equation (a) and (g) we find that R_1 and Q_1 are the same as R_2 and Q_2, if X_1 is playing the role of X_2 :
$R_1 = Y - A\ \bigcirc X_1$ from (a), now substitute X_1 by the value of X_2.

$$R_1 = Y - A\ \bigcirc(X_1 + a_1\ P_1)$$

But $A\ \bigcirc P_1 = S_1 \rightarrow R_1 = (prev.)R_1 - a_1 S_1$ which is the same expression suggested by equation (g).
On the other hand, in equation (a) we have

$$\begin{aligned} Q_1 &= A^* \bigcirc (Y - A\ \bigcirc X_1)\ \textit{substitute}\ X_2\ \text{for}\ X_1 \\ &= A^* \bigcirc (Y - A\ \bigcirc (X_1 - a_1 P_1)) \\ &= A^* \bigcirc (Y - A\ \bigcirc X_1 - a_1 A\ \bigcirc P_1) \\ &= A^* \bigcirc (R_1 - a_1\ S_1) \\ &= A^* \bigcirc R_2 \end{aligned}$$

which is the same value of Q_2 suggested by equation (g).
Since equations (a) and (g) perform the same computation, equation (g) can be deleted as verified above. The redundancy appears when step 1 follows step 3 only. Therefore, equation (g) is only needed when step 3 requests step 5 to be executed.

Numerical Example.

The following is a numerical example to demonstrate the correctness of the CG-algorithm version without looping in step 3 and after rearranging step 3 by itself in order not to repeat same computations in different steps, this version is attractive and suitable for the hardware parallelism of this algorithm.

Solving the least square problem

$$F=\frac{1}{2}\|A\,X-Y\|^2$$

where

$$A=\begin{bmatrix}1 & 1\\ 1 & 0\\ 1 & 0\end{bmatrix},\ A^*=\begin{bmatrix}1 & 1 & 1\\ 1 & 0 & 0\end{bmatrix},\ Y=\begin{bmatrix}1\\ 0\\ 2\end{bmatrix}$$

Then the solution of the normal equation

$$A^*A\,X=\begin{bmatrix}3 & 1\\ 1 & 1\end{bmatrix}X=A^*Y=\begin{bmatrix}3\\ 1\end{bmatrix}$$

is $X=\begin{bmatrix}1\\ 0\end{bmatrix}$. The outer-normal vectors to the two-dimensional constraint region (which is a unit square) are

$$W_1=\begin{bmatrix}-1\\ 0\end{bmatrix},\ W_2=\begin{bmatrix}0\\ -1\end{bmatrix},\ W_3=\begin{bmatrix}1\\ 0\end{bmatrix},\ W_4=\begin{bmatrix}0\\ -1\end{bmatrix}.$$

Start the algorithm with $k = 1$ and with $X_1=\begin{bmatrix}0\\ 0\end{bmatrix}$. Then $g_1(X_1)=0$, $g_2(X_1)=0$, $g_3(X_1)=-1$, $g_4(X_1)=-1$. Hence, the active index set is I = (1,2), which means that the H matrix is the null-matrix. The initial residual gradient vectors are, respectively,

$$R_1=\begin{bmatrix}1\\ 0\\ 2\end{bmatrix},\ Q_1=A^*R_1=\begin{bmatrix}3\\ 1\end{bmatrix}.$$

Enter 5: Minimize V with nonnegative λ_1, λ_2, where ,

$$V=\begin{bmatrix}3\\ 1\end{bmatrix}-\lambda_1\begin{bmatrix}-1\\ 0\end{bmatrix}-\lambda_2\begin{bmatrix}0\\ -1\end{bmatrix}=\begin{bmatrix}3+\lambda_1\\ 1+\lambda_2\end{bmatrix}$$

Hence, $\lambda_1=\lambda_2=0$, and $V=\begin{bmatrix}3\\ 1\end{bmatrix}$. Next consider

$$g_{1,1}+a[3\ \ 1]\begin{bmatrix}-1\\ 0\end{bmatrix}=-3\,a\le 0\rightarrow a>0$$

$$g_{2,1}+a[3\ \ 1]\begin{bmatrix}0\\ -1\end{bmatrix}=-a\le 0\rightarrow a>0$$

$$g_{3,1}+a[3\ \ 1]\begin{bmatrix}1\\ 0\end{bmatrix}=1+3\,a\le 0\rightarrow a\le\frac{1}{3}$$

$$g_{4,1} + a[3\ \ 1]\begin{bmatrix}0\\1\end{bmatrix} = -1 + a \le 0 \rightarrow a \le 1$$

Thus $a = 0.1$, and we leave step 5 with

$$X_1 = \begin{bmatrix}0\\0\end{bmatrix} + 0.1\begin{bmatrix}3\\1\end{bmatrix} = \begin{bmatrix}0.3\\0.1\end{bmatrix}.$$

Enter step 1 with this value of X_1 and compute

$$R_1 = \begin{bmatrix}1\\0\\2\end{bmatrix} - \begin{bmatrix}1 & 1\\1 & 0\\1 & 0\end{bmatrix}\begin{bmatrix}0.3\\0.1\end{bmatrix} = \begin{bmatrix}0.6\\-0.3\\1.7\end{bmatrix}$$

$$Q_1 = A^* R_1 = \begin{bmatrix}1 & 1 & 1\\1 & 0 & 0\end{bmatrix}\begin{bmatrix}0.6\\-0.3\\1.7\end{bmatrix} = \begin{bmatrix}2\\0.6\end{bmatrix}$$

$g_1 = -0.3$, $g_2 = -0.1$, $g_3 = -0.7$, $g_4 = -0.9$,

$I = \phi$ *or empty*, $H = I_2$.

Enter step 3

$$P_1 = H\,Q_1 = Q_1 = \begin{bmatrix}2\\0.6\end{bmatrix}$$

$$S_1 = \begin{bmatrix}1 & 1\\1 & 0\\1 & 0\end{bmatrix}\begin{bmatrix}2\\0.6\end{bmatrix} = \begin{bmatrix}2.6\\2\\2\end{bmatrix} \quad a_1 = \frac{4.36}{14.76} = 0.3$$

$$l_{1,1} = W_1^T P_1 = [-1\ \ 0]\begin{bmatrix}2\\0.6\end{bmatrix} = -2$$

$$l_{2,1} = W_2^T P_1 = [0\ \ -1]\begin{bmatrix}2\\0.6\end{bmatrix} = -0.6$$

$$l_{3,1} = W_3^T P_1 = [1\ \ 0]\begin{bmatrix}2\\0.6\end{bmatrix} = 2$$

$$l_{4,1} = W_4^T P_1 = [0\ \ 1]\begin{bmatrix}2\\0.6\end{bmatrix} = 0.6$$

$$X_2 = \begin{bmatrix}0.3\\0.1\end{bmatrix} + 0.3\begin{bmatrix}2\\0.6\end{bmatrix} = \begin{bmatrix}0.9\\0.28\end{bmatrix}.$$

$$g_{1,2} = g_{1,1} + a_1\, l_{1,1} = -0.9$$

$$g_{2,2} = g_{2,1} + a_1\, l_{2,1} = -0.28$$

$$g_{3,2} = g_{3,1} + a_1\, l_{3,1} = -0.1$$

$$g_{4,2} = g_{4,1} + a_1\, l_{4,1} = -0.72$$

$k = 1 \neq N$

$k = k + 1 = 2$; $X_1 := X_2$ and *go to step* 1

Enter step 1

$$R_1 = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0.9 \\ 0.28 \end{bmatrix} = \begin{bmatrix} -0.18 \\ -0.9 \\ 1.1 \end{bmatrix}$$

$$Q_1 = A^* R_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} -0.18 \\ -0.9 \\ 1.1 \end{bmatrix} = \begin{bmatrix} 0.02 \\ -0.18 \end{bmatrix}$$

$g_1 = -0.9$, $g_2 = -0.28$, $g_3 = -0.1$, $g_4 = -0.72$

$I = \phi$ *or empty*, $H = I_2$.

Enter step 3

$$P_1 = H\, Q_1 = Q_1 = \begin{bmatrix} 0.02 \\ -0.18 \end{bmatrix}$$

$$S_1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0.02 \\ -0.18 \end{bmatrix} = \begin{bmatrix} -0.16 \\ 0.02 \\ 0.02 \end{bmatrix} \quad a_1 = \frac{0.0328}{0.0264} = 1.25$$

$$l_{1,1} = W_1^T P_1 = [-1 \;\; 0] \begin{bmatrix} 0.02 \\ -0.18 \end{bmatrix} = -0.02$$

$$l_{2,1} = W_2^T P_1 = [0 \;\; -1] \begin{bmatrix} 0.02 \\ -0.18 \end{bmatrix} = 0.18$$

$$l_{3,1} = W_3^T P_1 = [1 \;\; 0] \begin{bmatrix} 0.02 \\ -0.18 \end{bmatrix} = 0.02$$

$$l_{4,1} = W_4^T P_1 = [0 \;\; 1] \begin{bmatrix} 0.02 \\ -0.18 \end{bmatrix} = -0.18$$

$$X_2 = \begin{bmatrix} 0.9 \\ 0.28 \end{bmatrix} + 1.25 \begin{bmatrix} 0.02 \\ -0.18 \end{bmatrix} = \begin{bmatrix} 0.925 \\ 0.055 \end{bmatrix}.$$

$$g_{1,2} = g_{1,1} + a_1\, l_{1,1} = -0.925$$

$$g_{2,2} = g_{2,1} + a_1\, l_{2,1} = -0.055$$

$$g_{3,2} = g_{3,1} + a_1\, l_{3,1} = -0.075$$

$$g_{4,2} = g_{4,1} + a_1\, l_{4,1} = -0.945$$

$k = N = 2$ *then*

$k := 1$

$$R_2 = R_1 - a_1 S_1 = \begin{bmatrix} 0.02 \\ -0.925 \\ 1.075 \end{bmatrix}$$

$$Q_2 = A^* R_2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.02 \\ -0.925 \\ 1.075 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.02 \end{bmatrix}$$

and go to step 5

Enter 5

$$I = \phi \rightarrow V = Q_2$$

Choose $a > 0$ such that

$$g_{1,1} + a\,[0.17 \;\; 0.02] \begin{bmatrix} -1 \\ 0 \end{bmatrix} = -0.925 - 0.17\,a \leq 0 \rightarrow a > 0$$

$$g_{2,1} + a\,[0.17 \;\; 0.02] \begin{bmatrix} 0 \\ -1 \end{bmatrix} = -0.055 - 0.022\,a \leq 0 \rightarrow a > 0$$

$$g_{3,1} + a\,[0.17 \;\; 0.02] \begin{bmatrix} 1 \\ 0 \end{bmatrix} = -0.075 + 0.17\,a \leq 0 \rightarrow a \leq 0.44$$

$$g_{4,1} + a\,[0.17 \;\; 0.02] \begin{bmatrix} 0 \\ 1 \end{bmatrix} = -0.945 + 0.02\,a \leq 0 \rightarrow a \leq 47.25$$

Thus $a = 0.1$, and we leave step 5 with

$$X_1 = \begin{bmatrix} 0.925 \\ 0.055 \end{bmatrix} + 0.1 \begin{bmatrix} 0.17 \\ 0.02 \end{bmatrix} = \begin{bmatrix} 0.942 \\ 0.057 \end{bmatrix}.$$

Enter step 1 with this value of X_1 and compute

$$R_1 = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0.942 \\ 0.057 \end{bmatrix} = \begin{bmatrix} 0.001 \\ -0.942 \\ 1.058 \end{bmatrix}$$

$$Q_1 = A^* R_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.001 \\ -0.942 \\ 1.058 \end{bmatrix} = \begin{bmatrix} 0.117 \\ 0.001 \end{bmatrix}$$

$g_1 = -0.942$, $g_2 = -0.057$, $g_3 = -0.058$, $g_4 = -0.943$

$I = \phi$ *or empty*, $H = I_2$.

Enter step 3

$$P_1 = H\,Q_1 = Q_1 = \begin{bmatrix} 0.117 \\ 0.001 \end{bmatrix}$$

$$S_1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0.117 \\ 0.001 \end{bmatrix} = \begin{bmatrix} 0.118 \\ 0.117 \\ 0.117 \end{bmatrix} \quad a_1 = \frac{0.0137}{0.0413} = 0.33$$

$$l_{1,1} = W_1^T P_1 = [-1 \;\; 0] \begin{bmatrix} 0.117 \\ 0.001 \end{bmatrix} = -0.117$$

$$l_{2,1} = W_2^T P_1 = [0\ -1]\begin{bmatrix}0.117\\0.001\end{bmatrix} = -0.001$$

$$l_{3,1} = W_3^T P_1 = [1\ 0]\begin{bmatrix}0.117\\0.001\end{bmatrix} = 0.117$$

$$l_{4,1} = W_4^T P_1 = [0\ 1]\begin{bmatrix}0.117\\0.001\end{bmatrix} = 0.001$$

$$X_2 = \begin{bmatrix}0.942\\0.057\end{bmatrix} + 0.33\begin{bmatrix}0.117\\0.001\end{bmatrix} = \begin{bmatrix}0.9808\\0.0573\end{bmatrix}.$$

$$g_{1,2} = g_{1,1} + a_1 l_{1,1} = -0.9811$$

$$g_{2,2} = g_{2,1} + a_1 l_{2,1} = -0.0573$$

$$g_{3,2} = g_{3,1} + a_1 l_{3,1} = -0.0194$$

$$g_{4,2} = g_{4,1} + a_1 l_{4,1} = -0.9427$$

$$k = 1 \neq N \rightarrow$$

$$k = k + 1 = 2,\ X_1 := X_2, \text{and } \textit{go to step } 1$$

Enter step 1 :

$$R_1 = Y - A X_1 = \begin{bmatrix}-0.038\\-0.98\\1.019\end{bmatrix}$$

$$Q_1 = A^* R_1 = \begin{bmatrix}1 & 1 & 1\\1 & 0 & 0\end{bmatrix}\begin{bmatrix}-0.038\\-0.98\\1.019\end{bmatrix} = \begin{bmatrix}0.001\\-0.038\end{bmatrix}$$

$$Q_1 \approx 0$$

Stop with solution :

$$X = \begin{bmatrix}0.9808\\0.0573\end{bmatrix}$$

The derived version of the conjugate gradient algorithm for constrained problems has been proven to be efficient and practical through theoretical reasoning and numerical support. The general pipeline architecture for the algorithm with optimal efficient steps could be implemented.

CHAPTER VIII

DIGITAL SIGNAL PROCESSING: REVIEW OF AVAILABLE HARDWARE AND APPLICATION NOTES TOWARDS THE DESIGN OF THE PARALLEL MACHINE

Jeff C. Treece

1. INTRODUCTION

Digital signal processing (DSP) chips are being used in an increasing number of applications. An obvious application of DSP's has been around for a while: real-time processing (e.g. FFT's and filters) of digital data that comes from an analog-digital (A/D) converter. For processing A/D data in this manner, the DSP need only operate on 16-bit (or less) precise fixed-point data, since common A/D converters have 8-16 bits of precision, but other uses for DSP chips exist, and often require features beyond these fixed-point DSP's. Today's DSP chip can be used as powerful microprocessor, complete with I/O and interrupt facilities, as well as a number cruncher for fixed or floating point numbers. The state-of-the-art in DSP chips is steadily changing, making possible many computing problems that were not possible only a short while ago. In our parallel machine design, we can fully exploit the dynamic DSP market, since the actual DSP used in the machine can be any of the advanced chips on the market. The machine is thus allowed to "grow" right along with the DSP market.

Most DSP's today are based on a pipeline architecture that typically includes multiple data paths internally and two or more data paths externally. The one-time limitation of a small addressable memory space no longer exists in recent DSP designs, with many DSP's being capable of directly addressing 16 megawords of RAM. Some DSP's contain an on-chip DMA controller so that The addressable memory can be loaded or off-loaded as the number crunching continues. Modern VLSI processors can often process data faster than today's "standard" memory chips can be accessed, so DSP's often have provisions for "slow" memory interfacing.

A recent trend has been to use DSP chips for applications that are more demanding in some ways, such as FFT's and convolutions of floating point (single or even double precision) arrays. We studied DSP chips in the latter context for this project: we investigated DSP's as number-crunchers for a scientific computing algorithm (conjugate gradient). This section of the report presents some of the DSP chips reviewed and suggests which of them would be useful for implementing the "conjugate gradient machine."

1 a. The DSP Market

DSP chips have been on the market for many years, but those available prior to about 1985 had too many limitations to be considered for many serious computing applications. For example, the AT&T WE-DSP32 was available in 1984, but could only directly address 56 kilowords of external memory, not enough for a large array of numbers. A more recent generation, the WE-DSP32C, can address 16 megawords, has a higher clock frequency, lower power requirements, has a hardware interrupt facility, and has many other improved features. Common today are DSP chips that have 32-bit wide data paths, and quickly operate on 16-bit fixed point numbers or 32-bit floating point numbers. DSP chips that operate directly on 64-bit double precision numbers have been slow to emerge, and at the time of this writing, only a couple exist. Double precision arithmetic can be done using single-precision DSP's, but it takes more time.

The major manufacturers of DSP chips (probably *AT&T*, *Motorola*, and *Texas* Instruments if we restrict the discussion to floating point chips) are on a fast-paced race to make the best DSP chip and claim a big section of the market. Texas Instruments has already announced a fourth generation in thier DSP family -- the third generation performs floating point math more than adequately for our machine design. Given the changing market, it makes sense to stick to fundamental designs that do not rely heavily on manufacturer-dependent features of a given DSP chip, and concentrating study on vendors that have shown a strong commitment to improving DSP technology.

1 b. A Look at the Available DSP Chips

Texas Instruments was one of the first serious manufacturers of DSP chips. With the Texas Instruments TMS310 DSP chip, modem, voice, and music product designers were armed with a new tool. Although TI set many industry standards in low-precision fixed point DSP chips, the same trend is not necessarily true in the late-generation DSP's: AT&T, Motorola, NEC, and several other manufacturers are presently producing 32-bit floating point DSP chips. The discussion in this report will concentrate on the TI, AT&T, and Motorola chips.

The three manufacturers use similar strategy in data flow: avoid bottlenecks by providing multiple paths for data. The prevalent emerging design is pipeline in nature, with various "DSP building blocks" along the pipe. [1] The "Harvard architecture" is most common (separate data and instruction busses). Ground-breaking prices can be quite high (e.g. $1300. each for a small quantity of TMS320C30, fall 1988), but as with any new technology, the prices drop rapidly as new products emerge. TI expects the TMS320C30 to be about $100. by 1990 (volume pricing). The following table summarizes some of the available DSP chips from TI and other manufacturers.

DSP Chips

Manufacturer	Chip	Description
Analog Devices	ADSP-2100	Fixed-point DSP that is reported to do a 1024-point complex FFT in approximately 6.6 ms.
Analog Devices	ADSP-3200	Family of floating-point components for DSP. Not a single-chip solution like most of the other products listed here. Can handle double presicion data types.
Texas Instruments	TMS32010	TI's first-generation DSP design: a fixed-point DSP that has found its way into modem, audio, and other 16-bit digital circuits.

[1] Often, the architecture seems more like a bus-connected architecture than a pipeline, however, multiple busses and multiple "DSP building blocks" means that several operations can be going on at once. The performance of such an architecture resembles a pipeline architecture.

DSP Chips		
Manufacturer	**Chip**	**Description**
Texas Instruments	TMS32020	TI's second-generation DSP design. This DSP is too limited for use in our proposed conjugate gradient algorithm. The biggest limitations are its 64k address space and lack of floating-point operations.
Texas Instruments	TMS320C30	TI's third-generation DSP design. The third-generation chip operates on floating point data, addresses 16M words of memory, and adds many new features.
Texas Instruments	TMS320C40	TI's fourth-generation DSP design. Announced, but no technical information available at this time.
Motorola	MC56000	RISC architecture DSP chip supporting fixed-point operations up to 32-bits wide. Harvard architecture, with two separate busses: a data bus and a program bus. Both busses can interface directly to memory, but typically use Motorola's special "Cache Memory Management Unit" (CMMU) interface chip.
Motorola	MC88000	RISC architecture DSP chip that supports single and double precision operations. directly addresses up to one gigaword of data and one gigaword of instruction; typically requires three chips (DSP and 2 cache management interface chips). Harvard architecture, with two separate busses as in the MC56000 design. Ideal chip for an application requiring limited DSP combined with some control (discussed in *Electronic Design*, April 28, 1988, p.39). Preliminary information; market status unknown.
Motorola	MC96000	Motorola's floating point extension of the MC56000. 26.7 MHz RISC architecture that is reported to do a 1024-point complex FFT in under 2ms. Handles single and single-extended precision operations (extended numbers have a 32-bit mantissa and 11-bit exponent). The chip directly addresses up to 4 gigawords of memory, but typically interfaced to memory via two special CMMU chips. Two separate memory spaces internally; ideal for complex numbers.
AT&T	WE-DSP32	32-bit floating point DSP. External memory limited to 56k words.
AT&T	WE-DSP32C	32-bit DSP for 16- and 24-bit integer operations and 32-bit floating point operations. 32-bit data bus and 24-bit (16M) address bus. Used in Pixel Machines' PXM 900 Series Graphics Workstations.
OKI	MSM6992	22-bit floating point (and 16-bit fixed point) single chip DSP. 100ns clock cycle; 2μm silicon gate CMOS; 64k word data space and 64k word program space. Slightly enhanced version, MSM699210, has more internal memory.

DSP Chips		
Manufacturer	**Chip**	**Description**
NEC	μPD77230	24- and 32-bit floating point DSP. 150ns instruction cycle; up to 13.4 MFLOPS

2. SELECTION OF A DSP FOR OUR APPLICATION

Our "parallel conjugate gradient machine" operates quickly by providing many computing elements, each capable of performing a small part of the conjugate gradient task. We assume that each computing element is composed primarily of a DSP and some memory. An "executive computing element," perhaps formed of a microprocessor rather than a DSP, is given the tasks of communicating, transmitting data, and scheduling the computing elements.

A few assumptions are necessary to insure that the machine operates efficiently. We assume that the problem assigned to the machine is highly computational, and not I/O bound, as our machine does little to improve I/O performance. We also assume that a DSP can spend a significant amount of time (compared to data transfer time) operating on local data. This assumption implies that each DSP computing element is equipped with a sufficient amount of local memory to hold the required data. These assumptions do not severely restrict the usefulness of the machine (see chapter VII); we are still able to address a very wide range of problems.

It has been predicted that in the near future we will see a 60 to 70% peformance increase per year in state-of-the art machines [E1]. New machines, such as the one we propose, should be flexible, both in terms of architecture and the technology used to build the machine. A design that is independent of a specific device technology allows the system to be upgraded to take advantage of the new technology. This consideration bears significance on the selection of a DSP: we prefer to select a DSP that is fairly general-purpose, but represents the leading edge of design.

2 a. Important Features

Among the top concerns in choosing a particular floating point DSP is the width of the address bus. A DSP that has a small memory space proves to be too severely limited; we require that each computing element can address enough data to hold a large array of numbers. We feel that a 16-bit (64k) wide address bus is too small, and that a 32-bit (4G) is probably more than we require (the extra space does not cause any particular problems). Many DSP chips have a 24-bit (16M) address bus, which is probably about right for our application.

We also preclude the use of fixed-point DSP chips. Fixed-only DSP's would be useful only if the algorithm were changed and scaled at each stage of the computation [K1] Double precision operations might simplify some steps, but we assume that we can implement the algorithm in single precision. If double precision is required, those

operations can be implemented in software. The DSP's that we consider are all 32-bit designs. This data size seems to be a rooted-in standard, and the data size determines the appropriate bus widths and data paths.

Other features assist in interfacing the DSP with the rest of the hardware of the machine. For example, Direct Memory Access (DMA) allows the DSP to operate somewhat independently of the memory interface, and is considered an important feature, but is not essential. Our design does much data transfer, and it would probably improve overall performance if each DSP could be freed from the mundane task of moving data from here to there. It is important to consider the availability of software and development tools for the DSP. We require at least a C, or other high-level compiler, and good technical support from the manufacturer. Fortunately, this has become a standard in the emerging DSP's. We will not concern ourselves with cost at this point for two reasons: we believe the application to be fairly cost-insensitive, and the DSP's studied are new, so their prices are constantly dropping.

Some important considerations are not addressed in this chapter but are covered in Chapter VII: bus design for sufficient bandwidth, amount and speed of local memory, number of computing elements, communication protocol, and implementation of the algorithm. We can compare DSP's on several different grounds: available features, speed of execution of particular code, cost, reputation of vendor (will it be upgraded?), and design considerations (how well does it fit into our machine?). The speed can be compared by looking at the speed of a particular bit of software, such as a complex FFT [2], as we know that this type of computation will be present in the algorithm. Such a comparison is of limited use since all of the considered DSP's execute such an algorithm in roughly the same amount of time, and the benchmark is very dependent on the particular implementation. Design considerations and available features, such as how the chip interfaces with memory, are more significant comparisons.

2 b. Comparison of Our Favorite Three

The three top choices for DSP's emerged early in our study; many of the available DSP's were not useful in our design because of limitations discussed above. We think that any of the three DSP's discussed in this section, Texas Instrument's TMS-320C30, AT&T's WE-DSP32C, and Motorola's MC96000, could be used in our design. We discuss tradeoffs involved when choosing one over another. Table 1 outlines some significant features of the three DSP chips.

The Motorola chip has two internal data spaces that make the chip ideal for executing code that operates on complex data. Since the Motorola chip has more internal data paths, it is a reasonable assumption that many problems could be executed more rapidly than on the other two DSP's. However, the standard design using the MC96000 uses

[2] Note that board-level products have been reported to do a 1024-pt complex FFT in less than a millisecond, for example, the Viper 8704, reported on p.22 of *Computer Design: News Edition*, January 16, 1989. This benchmark sheds light on the power of modern DSP chips: most of the *single-chip* DSP's discussed here do the same computation in approximately 2 ms.

	TMS-320C30	WE-DSP32C	MC96000
Harvard architecture	Yes	No	Yes
Data width	32 bits	32 bits	32 bits
Address width	24 bits (16M)	24 bits (16M)	32 bits (4G)
Clock speed	33.33M	40MHz, 50MHz	26.7MHz
Peak FLOPS	33M	25M	40M
Instructions/second	16.7M	12.5M	13.33M
1024-pt FFT time	1.67ms	?	<2ms
C Compiler	Yes	Yes	Yes
DMA controller	On-chip	On-chip for serial and parallel ports	Two On-chip
Technology	1.0μm CMOS	0.75μm CMOS	RISC
Internal RAM	2k words	1k words	1.5k words
Internal ROM	4k words	2k words	1k words
Instruction Cache	64 words	No	No
Serial port	two;8Mbit/s	16Mbit/s	sync. and async.
Parallel port	two	one;16-bit	32-bit interface
Interrupts	12;4 general purpose	4 internal; 2 external	3 external
Registers	28	22	10 96-bit
Available	Now	Now	Early 1989

two extra CMMU IC's to interface to memory, probably making the hardware more complex and costly.

All three chips have available C compilers for code development. The three DSP's have approximately the same number of registers available for accumulating results; the Motorola chip has fewer registers, but its registers can hold more data (96 bits). From the programmer's point of view, the three choices are probably almost identical. Most of the code development would be in C, and a small portion of the code would probably be done in assembly. For the assembly programming, a slight advantage might go to TI's and AT&T's DSP's due to flexibility of having a large number of smaller-sized registers and a simpler internal data flow. The instruction cache of the TI DSP would likely provide a speed-up in execution since each DSP (by the nature of the algorithm) executes the same code over and over. The lack of a cache in the MC96000 is probably made up for by its multiple internal busses, separate address spaces, and high degree of parallelism at the data transfer level.

Interface to memory seems most flexible in the TMS320C30. The TMS320C30 has two parallel ports and two serial ports for peripheral I/O and a general-purpose DMA controller for concurrent computation and data transfer. The chip has sufficient interrupt facilities for synchronizing to external events. All three DSP's considered are capable of addressing plenty of external memory; the 4 gigaword address space of the MC96000 is probably an insignificant advantage over the other two chips.

Overall, the TMS320C30 seems the best choice for our intended application. It executes instructions more rapidly than the other two, and seems to execute common

problems (e.g. 1024-pt complex FFT) at least as rapidly as the other two DSP's. Texas Instruments has been in the DSP business at least as long as anybody else, and has shown their commitment by continually improving on their designs. It appears that the next generation, the TMS320C40, is already available and might provide an immediate performance improvement. The cost of the TI part, though initially expensive ($1300.) will probably drop to $100. over the next two years.

2 c. Putting It All Together

Using the TMS320C30 as a benchmark, we can estimate the cost of the most costly hardware components of our proposed machine. We can not take into account development time, and we must estimate the cost of much of the hardware since the machine has not yet been designed. Let us assume that we will have one megaword of memory on each of the boards composing the system. The cost of the memory is a big part of the total cost since RAM chips are expensive as of December 1988; here we estimate the cost of RAM on today's market. TMS320C30 is capable of 60-ns instruction cycles, which is faster than the memory quoted below. Thus, to make full use of the chip, one could interleave, cache, or get (more expensive) faster RAM. Alternatively, the performance could be matched to the slow RAM via wait states. The price of the DSP quoted is expected to be $100. in volume by 1990. Table 2 estimates the cost of one computing element of our proposed machine. The estimated costs include only certain parts and exclude construction cost, miscellaneous chips, "central CPU" cost, and of course R&D. Note that a year ago, the 80ns RAM (1meg by 36 bits) would have been about $432. If indeed the DSP were to come down to $100 and memory costs were to fall back to their all-time low value, the hardware cost would change drastically ($632/board). We emphasize that the hardware cost is not the most significant consideration since for our proposed applications, the market is fairly cost-insensitive [S1], and in any case, the cost is not outrageous.

Item	Unit ($)	Total ($)
DSP Chip TMS320C30	1300.	1300.
80nS 1Mbit RAM chip	32.	1152.
Circuit board manufacture, much interface, PIA, TTL, buffer chips, and support circuitry; estimate		1000.
Total MINIMUM cost per board, estimated, based on today's market		3452.
Total MINIMUM cost, 50 boards		172,600.

3. BIBLIOGRAPHY

[A1] AT&T Microelectronics, DSP data book, "WE-DSP32C Digital Signal Processor," 1988.

[A2] Analog Devices, data manuals, "DSP Products Databook: DSP Microprocessors, Microcoded Support Components, Floating Point Components, Fixed Point Components," 1987.

[E1] "Computer System Architecture," *Electronic Design*, Vol. 37, No. 1, January 12, 1989, p.51.

[K1] "Computer Aided Implementation of Complex Algorithms on DSP's Using Automatic Scaling," *ICASSP*, Korina Kassapoglou and Martin Vetterli, 1987, p.1027.

[M1] Motorola, Inc, Motorola Literature Distribution, "BR282 and BR505, 56-bit General Purpose Digital Signal Processor," 1986.

[S1] Sabbagh Associates, Inc., market study for nondestructive evaluation hardware as part of Phase II SBIR project with the Department of Energy, contract number DE-AC02-83ER80096, 1986.

[T2] Texas Instruments, Inc, "Second-Generation TMS320 User's Guide," Houston, TX, 1987.

[T3] Texas Instruments, Inc, "Third-Generation TMS320 User's Guide," Houston, TX, 1988.

CHAPTER IX

IMAGE PROCESSING OF EDDY CURRENT DATA

Bishara Shamee

1. Overview

In this chapter we summarize the processing developed for the analysis of eddy current images. One the most useful scheme was the segmentation of images used to obtain the classifier parameters in Chapter VII. In addition, the statistical properties of the flaw and background regions were used to obtain an estimate of the regularization parameter which preconditions the inversion data.

2. Image Processing for Flaw Detection

In this section, applicable techniques of image processing will be explored in order to isolate flaw regions, and study the statistical properties of the flaws and background. The terminology adopted in computer vision will be used here. Laboratory measurements will be denoted by images, and each sample by a pixel. The data collected from an experiment is quantized in the spatial domain, with resolution bounded by the precision of the data acquisition system.

Initially, laboratory eddy current measurements are represented by images. The images have eight bits of resolution converted linearly according to:

$$I(x,y) = \frac{255}{m_1 - m_0} [\, d(x,\, y) - m_0] \tag{1}$$

where $d(x,y) =$ is the pixel intensity at $(x,\tilde{y})$, $m_1 = \max\{d(x,\, y)\}$, $m_0 = \min\{d(x,\, y)\}$, and $I(x,\, y)$ is the resulting image. Variations to this conversion rule produce different visual results. Different conversion schemes have not been investigated, in particular non-linear conversions such as Logarithmic scaling.

In chapter VI, the statistical properties of the flaw and background regions were needed. The process of separating the flaws from the background is known in computer vision area by *segmentation*. Segmentation is the process of identifying homogeneous regions of the image, as covered in the next section.

2.1. Segmentation of Eddy Current Measurements

Results of segmentation will be used to estimate the statistical properties of the flaws and background, and to obtain an estimate of the flaw support for inversion. Segmentation differs from statistical detection since it does not yield statistics about the flaw, and there does not exist an adequate measure of quality. Most segmentation results are evaluated visually, and various segmentors yield different results when applied to the same class of images. Hence, for a given class of images there may exist a "good" segmentor. The results, if interpreted from detection point of view, have a high false alarm error rate. Specifically, segmentation may not detect the absence of a flaw.

In our case, segmentation will be used as a preprocessing step, with the possibility of using the results for inversion. The main utility of the segmentation is to obtain data that gives sufficient characteristics about the classes. Segmenting large number of measurements would reduce the estimation error of the statistical parameters of the flaws and background regions.

The segmentation algorithm devised here for the purpose of flaw region separation is based on the fact that the amplitude response of eddy current to flaws are higher than those to the

background. The segmentor devised here, seems to perform well on laboratory data for almost all types of experimental arrangements. A threshold is selected from the histogram of the image. Since the flaw regions are those at the high end of the range axis, the threshold is selected with the following method.

Suppose that $h(i)$, (i = 0, 1, . . ., n) is the histogram of an image. Denote the first moment by μ_1, and the second moment by μ_2. The threshold is selected at:

$$T = \mu_1 + 0.5\sqrt{\mu_2} \tag{2}$$

where

$$\mu_1 = \sum_{i=0}^{n} i\, h(i) \quad \text{and} \quad \mu_2 = \sum_{i=0}^{n} i^2\, h(i) - \mu_1^2. \tag{3}$$

The results of segmenting the satin weave is shown in Figure 1. The reader may wish to refer to the FOURTH QUARTERLY REPORT of this contract for more segmentation results of eddy current data.

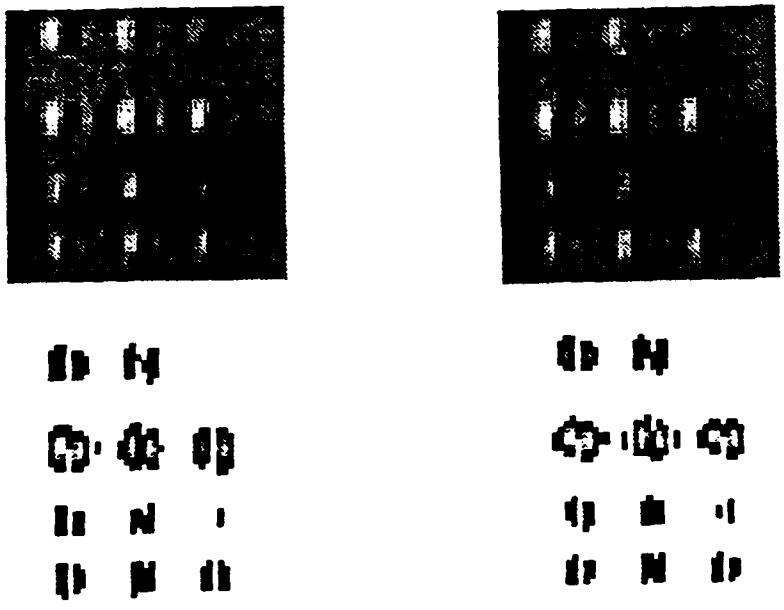


Figure 1: Segmantation of laboratory data of the drilled satin weave sample. (a) Original real part. (b) Original Imaginary part. (c) Segmented real part. (d) Segmented imaginary part.

2.2. Planar Filtering of Eddy Current Images

Some measurements made in the laboratory reveal a planar offset that made segmentation of the images difficult. Segmentation could be improved if the planar offset is removed from the data. This preprocessing step is necessary because the planar offset is due to equipment setup and is not a characteristic of eddy currents.

Let $D(x,y)$ be the measurement data or image over rectangular grid in the measurement plane. In addition, this offset may yield incorrect inversion results. $\{ (x,y): 0 \le x < M, 0 \le y < N \}$. The form of the planar offset over the grid $\{ (x,y): 0 \le x < M, 0 \le y < N \}$ is given by:

$$z = a\,x + b\,y + c. \tag{4}$$

The problem now is to determine a, b, and c such that the least square error is minimized, where the least square error is defined by:

$$\varepsilon = \sum_{(x,y)\in UxV} [\,D(x,\,y) - (a\,x + b\,y + c\,)\,]^2, \tag{5}$$

where U and V are symmetric windows. Upon taking partial derivatives with respect to a, b, and c, we have:

$$a \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} x_i y_j + b\,m \sum_{j=0}^{n-1} y_j^2 + c\,m \sum_{j=0}^{n-1} y_j = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} y_j\,D(x_i,y_j) \tag{6}$$

$$a\,n \sum_{i=0}^{m-1} x_i^2 + b \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} x_i y_j + c\,n \sum_{i=0}^{m-1} x_i = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} x_i\,D(x_i,y_j) \tag{7}$$

$$a\,n \sum_{i=0}^{m-1} x_i + b\,m \sum_{j=0}^{n-1} y_j + c\,nm = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} D(x_i,y_j) \tag{8}$$

using the properties of symmetric sets, the plane parameters can be obtained without any matrix inversion. Solving for a, b, and c we obtain,

$$a = \frac{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} x_i\,D(x_i,y_j)}{n\sum_{i=0}^{m-1} x_i^2}, \quad b = \frac{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} y_j\,D(x_i,y_j)}{m\sum_{j=0}^{n-1} y_i^2}, \quad c = \frac{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} D(x_i,y_j)}{mn}. \tag{9}$$

Each pixel of the image is then replaced by $D(x,\,y) - z(x,\,y)$ Note that the segmentation results have been improved as shown in Figure 2. A final remark, the planar filter constructed, will not affect the data if the offset is not large. Hence, this filtering step can be performed on all the images without degrading the measurements.

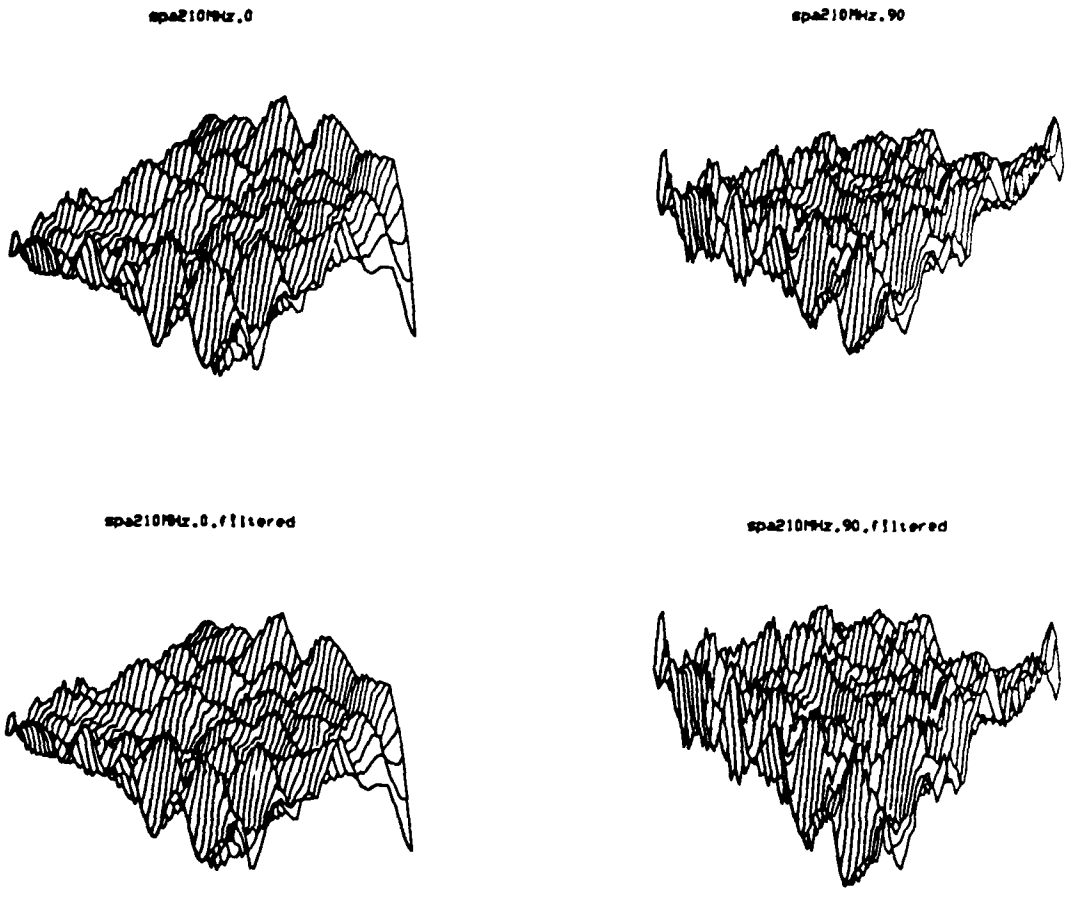


Figure 2: Planar offset extraction. (a) Original real part. (b) Original Imaginary part. (c) Filtered real part. (d) Filtered imaginary part.

2.3. Edge Detection of Flaw Measurements

In intensity visual images, the edges can be used to detect variations in the intensity. These variations could be used to outline the boundaries of regions in images. In eddy current measurements, the variations in amplitude response indicate changes. These changes are attributed to the presence of the flaw. Noise also contribute to these variations, however, flaw regions can be separated from noisy samples. Edge detection can be used here to enhance the visibility of the flawed regions. Edge detection techniques are standard in the area of computer vision, and the reader is referred to [H1] for more details.

2.3.1. Sobel Edge Detection

Perhaps Sobel edge detection is the easiest edge detector. It is utilized heavily as an initial investigation for vision systems. Sobel edge detection is basically a convolution of horizontal and vertical masks over the image. Since variations are detected by differentiation, the masks used are an approximation to the directional derivative. The result is a gradient field with

magnitude and direction over the image. The magnitude indicate the strength of the edge, while the direction is the direction of the gradient field. The actual edge direction is offset by 90 degrees from the gradient direction. The masks are:

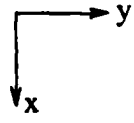
Horizontal Mask (D_y)

	-1	0	1
1/4	-2	0	2
	-1	0	1

Vertical Mask (D_x)

	1	2	1
1/4	0	0	0
	-1	-2	-1

It is assumed that the scanning of the image is done from top to bottom, right to left as in a raster scan. The coordinate system used is:



The convolution of the masks with the image yields an approximation of the gradient in the x and y directions. The edge strength at a pixel is defined by:

$$D = (D_x^2 + D_y^2)^{1/2} \tag{10}$$

with corresponding orientation:

$$E_\theta = \tan^{-1}(D_y / D_x). \tag{11}$$

Figure 5 shows the result of the Sobel edge detector.

2.4. Iterative Thinning of Edges: Eberlein's Thinning

In this section, an iterative thinning algorithm will be applied. Results of thinning are used to localize edges. Furthermore, thinning could be used to extract the orientation of the fibers in composite material.

The thinning algorithm described here is an iterative process. Originally, in [E1], the algorithm is used with four principal directions: North, South, East and West. The algorithm have been extended to thin along eight principal directions as discussed next.

The gradient of the image is taken. To each edge pixel an orientation and a strength are assigned. Denote the orientation of the central edge pixel by the principal direction. A 3x3

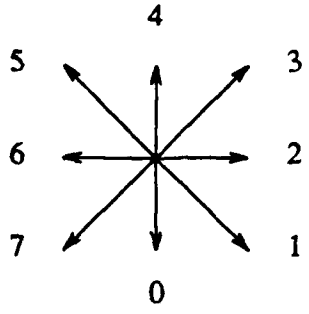


Figure 3: Quantization of edge orientation into eight directions.

moving window is passed over the edge image, the two neighbors at right angles with the principal direction are compared with the central pixel.

5	4	3
6		2
7	0	1

Figure 4: Neighbor labeling for a 3x3 window.

Table 4 shows the corresponding neighbors for each orientation.

Table 4: Neighbors used in thinning eight directions.

Neighbors associated with each principal direction.		
Direction	Neighbors	
0	2	6
1	3	7
2	4	0
3	5	1
4	6	2
5	7	3
6	0	4
7	1	5

If any neighbor used in the comparison is less than the central pixel, then the central pixel is increased by a portion of that neighbor, and the neighbor is decreased by that amount. Nothing happens if the neighbor is greater than or equal to the central pixel. if $0 < \alpha < 0.5$ denotes the absorption constant, then the central pixel is incremented by α times the strength of the neighbor. Mathematically, let p be the central pixel and n be the neighbor at right angles with the same orientation as p, the thinning process is then:

$$p \leftarrow \alpha\, u(p-n) \tag{12}$$

$$n \leftarrow n - \alpha\, u(p-n) \qquad \text{where } u(.) \text{ is a step function.} \tag{13}$$

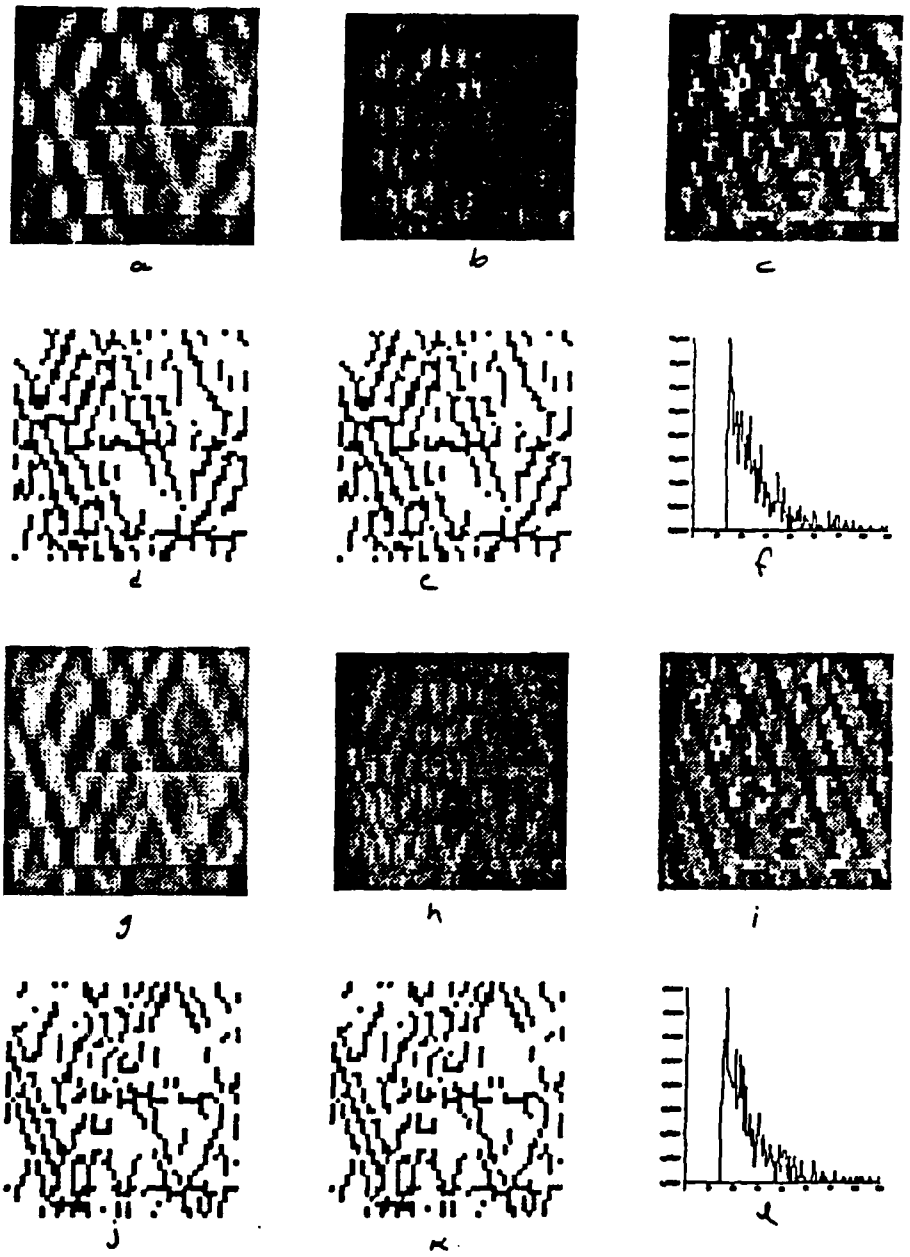


Figure 5: Sobel edge detection and thinning. (a) Original real part (b) Sobel edge strength of the real part (c) Phase angle of real part (d) Real part thinned (e) Phase thinned (f) Histogram of the edges (g) Original imaginary part (h) Sobel edge strength of the imaginary part (i) Phase angle of imaginary part (j) Imaginary part thinned (k) Phase thinned (l) Histogram of the edges

2.5. Spatial Filtering of Tow Signal

In this section filtering of the tows from laboratory measurements will be presented. The filter utilized here is simple in nature and evaluates the applicability of the Fourier transform in tow elimination.

Tow elimination can be described by eliminating the spatial frequencies with large amplitude. For example, Figure 6 shows an image with both a flaw and oscillation due to the presence of tows that is a characteristic of the material under examination. If these measurements are inverted, then the tows would be identified as flaws. By definition of flaws, they imply a change in the conductivity of the material. Even though this is true in the strict sense, it is not desired to reconstruct the tows in the material. Thus their elimination prior to reconstruction is necessary.

The straight forward approach to tow elimination from the measurements is to implement a spatial filter whose response matches that of the tows. However, the flaw regions may be affected by the filtering. In particular, when the flaws have a frequency structure close to that of the tows, then the filter will degrade the flaws as well. This point will be examined more closely after the filter is presented in the next section.

2.5.1. Spatial Filter Design by Magnitude Thresholding

As mentioned above, a simple filter can be designed by considering the frequencies with large amplitude. In order to extract these frequencies, the fourier transform is extracted and the magnitude response is obtained. Once the magnitude response of an image is known, one may look at these frequencies that have large amplitudes and suppress them. This is done here except that the algorithm used for detecting large frequencies is based on the histogram of the magnitude of the fourier transform. To extract the large frequencies, the segmentation algorithm used for flaw extraction will be utilized here. The segmentation algorithm is based on the amplitude of the measurements and when considering the magnitude response as an image, the desired frequencies are extracted or suppressed based on the desired components. Segmenting the amplitude response yields the support of the frequencies with large amplitude. The filter is then defined over this support.

let I(x, y) be a given meaurement and $\hat{I}(\omega_x, \omega_y)$ be its corresponding fourier transform. Denote ω_o to be the set of frequencies obtained from segmenting the magnitude of the fourier transform. Then ω_o characterize the response of the tows due to their periodicity. The filter is then:

$$H(\omega_x,\omega_y) = \begin{cases} 1, & (\omega_x,\omega_y) \in \Omega_o \\ 0, & 0. \end{cases}$$

The impulse response of the filter is given by the inverse fourier transform of the filter frequency response. A sample image containing a flaw and tow signal is processed in Figure 6. The sequence of images and plots show the various stages of the processing in order to eliminate the stripping.

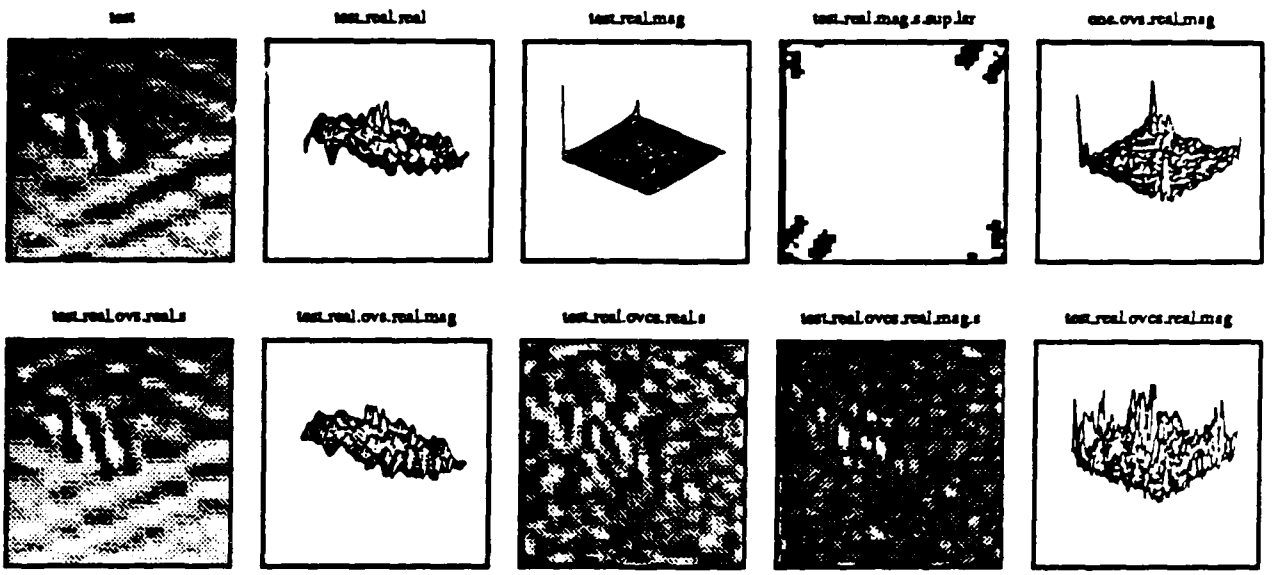


Figure 6: Processing sequence of tow elimination. (a) original image with tows and flaw, (b) three-dimensional plot of the original image, (c) the magnitude of the fourier transform of the image, (d) the significant frequencies (dark regions), (e) the impulse response of the filter, (f) the filtered image over the support of the significant frequencies, (g) three-dimensional plot of the stripped part, (h) the filtered image over the complement of the significant frequencies, (i) the magnitude of the filtered image, (j) three-dimensional plot of the magnitude.

3. LM Parameter and Noise Filtering

In this section, noise in eddy current measurement will be characterized and preprocessing the data will be studied along with obtaining estimates of the LM parameter. The pre-processing will then consist of smoothing the data for inversion in particular to obtain bounds on the regularizing parameter. Most of the available smoothing techniques require knowledge of the noise properties. The term *noise* will denote the random component of the measurement. It is is introduced at various stages of the measurement process and it is difficult to derive its distribution.

Starting with the data acquisition, noise enters the system from many sources. In order to characterize the noise, the following is assumed throughout this section:

(1) The noise in the system does not depend on the material under examination.

(2) The noise does not depend on the location of the sensor or time.

(3) The data acquisition system has sufficient bit resolution.

Assumption (1) implies that the noise properties are independent of the sample work piece (i.e. satin weave, foil target,...etc.). For example, the distribution and the first and second moments of the noise will not change when the sample changes. Furthermore, the tows which will be eliminated for reconstruction, are not noise because their response is deterministic and not random. Assumption (2) implies that the noise is stationary with respect to the spatial location of the sensor or the time origin. Assumption (3) implies that the quantization error will be ignored. All these assumptions will lend themselves to a mathematical formulation in smoothing the data.

Suppose a signal f is to be measured. The measurement process introduces additional noise to the present noise in the system making the estimate of the signal necessary. In relation to inversion, the workpiece response is measured, and it is desired to estimate the "true" response of the workpiece. If the assumption of additive noise is to hold, the following block diagram (model) results:

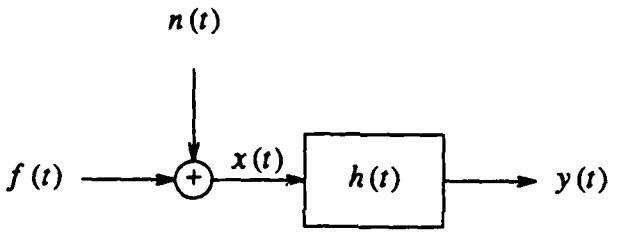


Figure 7: Underlying model of the measurement and "true" signal

The true signal $f(t)$ is the ideal measurement with no noise, $n(t)$ is the additive noise introduced by the system dynamics or measurement, $x(t)$ is the measured signal, $h(t)$ is the smoothing filter to be designed, and finally $y(t)$ is the *best* estimate of $f(t)$. The following two sections will cover the smoothing of the measured signal. The measurements have been segmented in order to obtain the noise and signal characteristics. The spectral properties of each component will be needed to design the filter. Segmentation would give an estimate of these properties since it separates the measurement into two regions. We should mention that the segmentation results for both regions contain noise. The region identified with flaws should contain most of the flaw signal, and the region identified with background should contain mostly noise. The segmentation should give sufficient data to estimate the spectral properties of the noise and the flaw regions. These properties are then used to derive a smoothing filter as shown in the next two sections. First filter will be based on the Frequency domain, while the second is based on the spatial domain and is more adequate to solve on a digital computer.

3.1. Smoothing Filter Design -- Frequency Domain [P1]

The spectral properties of the model can be used to derive a smoothing filter. The model equations are:

$$x(t) = f(t) + n(t), \tag{14}$$

$$y(t) = x(t) * h(t). \tag{15}$$

Substituting Equation (14) in (15) the following holds:

$$y(t) = (f(t) + n(t)) * h(t) \equiv y_f(t) + y_n(t) \tag{16}$$

where $y_f(t) = f(t) * h(t)$ and $y_n(t) = n(t) * h(t)$ are the responses to each component. In practice such a decomposition is not possible since only the measurement $x(t)$ is available. The *signal to noise ratio* at time t_o is defined by:

$$\rho(t_o) = \frac{|y_f(t_o)|}{\sqrt{E[|y_n(t_o)|]^2}}, \tag{17}$$

Note that this expression assumes knowledge of the signal and noise characteristics. The segmentation results may provide adequate properties to compute the signal to noise ratio or even to find the smoothing filter $h(t)$. Considering the square of the denominator in Equation (17) above we have,

$$E[y_n(t_o)] = \frac{1}{2\pi} \int_{-\infty}^{+\infty} S_{nn}(\omega)\, H(\omega)\, H^*(\omega)\, d\omega \tag{18}$$

where $H(\omega)$ is Fourier transform of the impulse response of $h(t)$. The numerator is

$$y_f(t_o) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} F(\omega)\, H(\omega)\, e^{j\omega t_o}\, d\omega, \tag{19}$$

where $F(\omega)$ is the Fourier transform of the signal $f(t)$. Note that only the sum of the signal $f(t)$ and the noise is measured. The segmentation may yield accurate estimates of these terms, however, the basis for the accuracy has to determined by comparing the inversion results with actual known cases. This numerator can be estimated by using *Schwarz'* inequality, that is,

$$\left| \frac{1}{2\pi} \int_{-\infty}^{+\infty} F(\omega)\, H(\omega)\, e^{j\omega t_o}\, d\omega \right|^2 \le \int_{-\infty}^{+\infty} \frac{|F(\omega)|^2}{S_{nn}(\omega)}\, d\omega \int_{-\infty}^{+\infty} |S_{nn}(\omega)\, H(\omega)|^2\, d\omega. \tag{20}$$

The signal to noise ratio $\rho(t_o)$ is bounded by:

$$\rho(t_o) = \frac{|y_f(t_o)|}{\sqrt{E[|y_n(t_o)|]^2}} \leq \frac{1}{2\pi} \int_{-\infty}^{+\infty} \frac{F^*(\omega) F(\omega)}{S_{nn}(\omega)} d\omega. \tag{21}$$

The signal to noise ratio is maximum if the previous equation is an equality, hence,

$$H(\omega) = k \frac{F^*(\omega)}{S_{nn}(\omega)} e^{j\omega t_o}. \tag{22}$$

The impulse response of the filter $h(t)$ is the inverse Fourier transform of Equation (22). The maximum signal to noise ratio is:

$$\rho_{max} = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \frac{F^*(\omega) F(\omega)}{S_{nn}(\omega)} d\omega. \tag{23}$$

The filter given in Equation (22) is well known and is called a *Matched Filter*. The *Wiener* filter is covered next for comparison with the matched filter. The Wiener filter will be derived by using the *Orthogonality Principle*. The expression for $y(t)$ can be written in terms of the measured signal $x(t)$ to give,

$$\int_{-\infty}^{+\infty} x(t-\alpha) h(\alpha) d\alpha, \tag{24}$$

where $h(t)$ is the impulse response of the desired filter to be determined by minimizing the total error,

$$e = E[(f(t) - y(t))^2]. \tag{25}$$

By using the orthogonality principle (see [P1]), the resulting equation is:

$$E\left[\left[f(t) - \int_{-\infty}^{+\infty} x(t-\alpha) h(\alpha) d\alpha\right] x(t-\tau)\right] = 0, \quad \text{for } all\ \tau. \tag{26}$$

In terms of the correlation functions we have:

$$R_{fx} = \int_{-\infty}^{+\infty} R_{xx}(\tau-\alpha) h(\alpha) d\alpha \quad \text{for } all\ \tau. \tag{27}$$

Solving Equation (27) for $h(t)$ yields the Wiener filter. Note that this equation involves the cross correlation of the signal and the noise R_{fx}, and the auto-correlation of the measurement R_{xx}. The error that results from using the Wiener filter is given by:

$$R_{ff}(0) - \int_{-\infty}^{+\infty} R_{fx}(\alpha)\, h(\alpha)\, d\alpha. \tag{28}$$

If the noise and the signal are orthogonal, then the filter response reduces to:

$$H(\omega) = \frac{S_{ff}(\omega)}{S_{ff}(\omega) + S_{nn}(\omega)}, \tag{29}$$

a very well known result.

3.2. Smoothing Filter Design -- Spatial Domain [A2]

Again by assuming the signal model in Figure 19, a smoothing filter is examined next without the transform method. The frequency methods require spectral estimation which by itself is an involved subject. An important point is that the noise characteristics are difficult to obtain analytically and thus the only method of obtaining these properties is based on the segmentation.

Suppose the noise is small compared with the signal, or even suppose that the noise is ignored. Then the actual measurement would be used as an estimate of the true signal. Thus if $x(t)$ is an estimate of $f(t)$ then the expected value of the error squared is:

$$E[(f-\hat{f})^2] = E[(f-x)^2] = E[(f-f-n)^2] = E[n^2] = \sigma_n^2, \tag{30}$$

where the noise is assumed to be zero mean process. Equation (30) above implies that if the noise is ignored, then the expected error is equal to the variance in the noise. This is unacceptable under some actual data noisy measurements. Assuming the signal and the noise are Gaussian, with joint distribution would lead to better estimate. First, the Gaussian assumption is examined, then the question of what happens to the error if this assumption fails.

Suppose the signal $f(t)$ and the noise $n(t)$ are zero mean Gaussian processes with variances σ_f^2 and σ_n^2 repectively. Consider the conditional density $\delta_{f|x}$, the density of the signal f given the measurement x to be:

$$\delta_{f|x}(f|x) = \frac{1}{(2\pi)^{1/2}} \exp\left|-\frac{1}{2}\,\frac{(f-\mu_{f|x})^2}{\sigma_{f|x}^2}\right| \tag{31}$$

where $\mu_{f|x}$ and $\sigma_{f|x}$ are defined by:

$$\mu_{f|x} = \frac{x\,\sigma_f^2}{\sigma_f^2 + \sigma_x^2}, \tag{32}$$

$$\sigma_{f|x} = \frac{\sigma_f^2\,\sigma_f^2}{\sigma_f^2 + \sigma_x^2}. \tag{33}$$

Thus if the estimate of the signal f is taken to be the conditional expectation of the signal given the measurement, i.e.,

$$\hat{f} \equiv E\,[\,f \mid X = x\,] = \mu_{f|x} = x\,\frac{\sigma_f^2}{\sigma_f^2 + \sigma_x^2}, \tag{34}$$

then the expected square error is:

$$E\,[\,(\hat{f} - f)^2\,] = \sigma_{f|x} = \frac{\sigma_f^2\,\sigma_f^2}{\sigma_f^2 + \sigma_x^2} < \min\,(\,\sigma_f^2,\,\sigma_x^2). \tag{35}$$

meaning that on the average, a better estimate is obtained.

3.3. LM Parameter Estimation

The laboratory measurements are complex numbers defined over a rectangular grid in the spatial domain. The LM Parameter depends on the statistical parameters of the two-dimensional measurements which will be denoted by γ, defined in [K1] as:

$$\gamma = \frac{\sigma_n^2}{\sigma_a^2}, \tag{36}$$

where σ_n^2 is the noise variance, and σ_a^2 is the signal variance. The noise is assumed to be independent and additive to the signal. Further more, the signal and the noise are uncorrelated stochastic processes [K1]. If the correlation assumption fails, then the expression for the LM parameter involves the correlation matrices of the signal and noise processes. If the uncorrelated assumption does not hold, then explicit expressions have to imbedd the correlation into the inversion scheme. At this time, the γ is evaluated as given above in Equation (?). Should this expression prove incorrect, then the signal and noise have to be treated as correlated random variables.

The expression of γ assumes that the noise and signal properties are known. In earlier reports, a method for segmenting the eddy current measurements have been presented. The segmentation results will be used to estimate the γ. To review, segmentation yields two regions, a flaw region and a background region. These regions are not exact since noise is contained in both of them. If the sample size is large enough, then our estimate of the γ should be accurate.

The mean and variance of a complex random variable is computed next for later reference. Let z be a complex random variable. That is, z has the form $z = x + j\,y$ where j is the imaginary unity in the complex plane, x and y are real random variables. If x and y are stochastic processes, then z is called a *complex stochastic process*. A complex random variable is the sum of two real random variables with j being a constant. The properties of interest are the mean and the variance of a complex random variable. The expectation of a complex random variable is:

$$\mu_z \equiv E\,[z] = E\,[x + j\,y] \tag{37}$$

By linearity of the expectation operator, the expected value reduces to:

$$\mu_z \equiv E\,[z] = E\,[x] + j\,E\,[y] = \mu_x + \mu_y. \tag{38}$$

The variance of a complex random variable is obtained from the definition:

$$Var\,(z) \equiv E\,[(z - \mu_z)\,(z - \mu_z)^*], \tag{39}$$

where * denotes complex conjugation. Expanding the terms above, one obtains the following expression for the variance:

$$Var\,(z) = E\,[(z - \mu_z)(z^* - \mu_z^*)] \tag{40}$$

$$= E\,[z\,z^* - z\,\mu_z^* - \mu_z\,z^* + \mu_z\,\mu_z^*] \tag{41}$$

$$= E\,[z\,z^*] - \mu_z^*\,E\,[z] - \mu_z\,E\,[z^*] + \mu_z\,\mu_z^* \tag{42}$$

$$= E\,[|z|^2] - |\mu_z|^2. \tag{43}$$

The magnitude of the complex random variable z is expressed in terms of its real and imaginary components (i.e. $|z|^2 = x^2 + y^2$) to yield:

$$Var\,(z) = E\,[x^2 + y^2] - \mu_x^2 - \mu_y^2, \tag{44}$$

$$Var\,(z) = (E\,[x^2] - \mu_x^2) + (E\,[y^2] - \mu_y^2) \tag{45}$$

$$Var\,(z) = Var\,(x) + Var\,(y) = \sigma_x^2 + \sigma_y^2 \tag{46}$$

That is, the variance of a complex variable is the sum of the variance of the real part and the variance of the imaginary part. Hence, the final expression for γ is then:

$$\gamma = \frac{\sigma_{n_r}^2 + \sigma_{n_i}^2}{\sigma_{a_r}^2 + \sigma_{a_i}^2}. \tag{47}$$

The segmentation results of the actual data set (the actual real and imaginary measurement) is in Appendix A. The dark regions are those points identified as flaws, the remaining points are identified as background. For each frequency, the variances are computed and the ratio γ profile is shown in Figure 20. Table 2 lists some of the properties of γ.

Table 2: Properties of γ for some experimental data sets.

Some Characteristics of γ				
Set	Average μ_γ	Variance σ_γ^2	Minimum	Maximum
dec01	0.46233	0.001448	0.322163	0.50251
dec07	0.16151	0.002750	0.009524	0.09224
nov20	0.41857	0.000813	0.378043	0.50224

Should this method provide an inadequate value for γ, then the alternative is to pre-process the data with any of the two filters derived earlier.

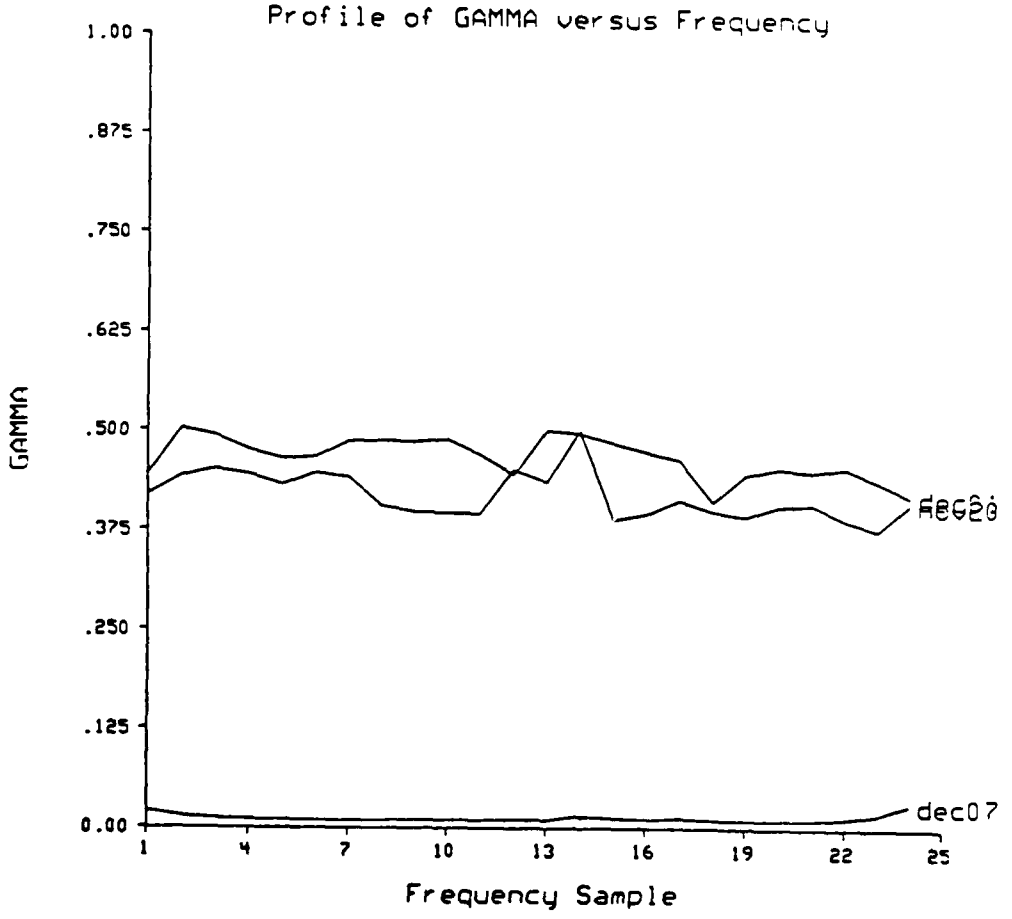


Figure 8: LM parameter of some actual data.

BIBLIOGRAPHY

[A1] Aho, A., Hopcroft, J., Ullman, J., *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Company, 1974.

[A2] Anderson, B., Moore, J., *Optimal Filtering*, Prentice Hall, 1979.

[F1] Fukunaga, K., *Introduction to Statistical Pattern Recognition*, Academic Press, 1972.

[G1] Gibbons, Alan *Algorithmic graph Theory*, Cambridge University Press, 1985.

[H1] Horowitz, E., Sanhi, S., *Fundamentals of Data Structures in Pascal*, Computer Science Press, 1982.

[K1] Kawata, S. and Nalcioglu, O., *Constrained Iterative Reconstruction by the Conjugate Gradient Method*, IEEE Transactions On Medical Imaging, Vol. MI-4, No. 2, June 1985, pp 65-71.

[P1] Papoulis, A., *Signal Analysis*, McGraw-Hill, 1977.

[P2] Papoulis, A., *Probability, Random Variables and Stochastic Processes*, McGraw-Hill, 1981.

[W1] Wong, E. Hajeck, B., *Stochastic Processes in Engineering Systems*, Springer-Verlag, 1981.

CHAPTER X

DESIGN OF OPTIMAL SYSTOLIC ARRAYS

by FOUAD MRAD

1. Introduction

The *conjugate gradient method* is a very famous method in optimizing a given objective function with or without constraints. Parallelism in various steps of this algorithm is clear and the recurrence phenomenon is obvious. Our objective is to exploit the parallelism in this method, and design *Optimal Systolic Arrays* which represent different steps as a set of recurrence equations with minimum Space and Time complexities. Parallel computing is receiving a rapidly increasing amount of attention. In theory, a collection of processors that operate in parallel can achieve substantial speedups. In practice, technological developments are leading to the actual construction of such devices at low cost. Many architectures for parallel computations have been proposed in the literature. Some of these machines actually exist or are being built, others are useful for theoretical design and analysis of parallel algorithms while their realization is not feasib le due to physical limitations [3].

The quality of the parallelism will be judged by the resulting *speed up* , which is the running time of the best sequential implementation of the algorithm divided by the running time of the parallel implementation using *p processors* ,and the *processor utilization* ,which is the speed up divided by *p*. The best that one can hope to achieve is a speed up of *p* and a processor utilization of one [3], [4].

There are a number of reasons why parallel implementation can be slower than anticipated in a theoretical analysis, slower perhaps than the fastest sequential implementation. Therefore the goal will be the design of an *optimal implementation* with due attention for:
processor structure,communication costs, and data distribution.

2. On Supercomputing with Systolic Array Processors

The main driver in designing such processors is the need for low cost, high density, fast processing devices.
Current parallel computers can be classified into three structural classes : *Vector Processors, Multi-Processor Systems, and Array Processors.* The focus has been on the last class because of promissing solutions to our need. An *Array Processor* is an array of processor elements (PE) with direct (*Static*) or indirect (*Dynamic*) interconnection. The most critical issue is communication or moving data between PE's in large scale interconnection network. *Signal Flow Graph* is the most useful graphical representation for scientific and signal processing computations. We find

two major classes of SFG : Those with local interconnections, and those with global ones.

Now, we can define Systolic Processors. *Systolic Processors* are a new class of *pipelined* array architecture. In general, a systolic system is a network of processors which rhythmically compute and pass data through the system. It has the following properties : *Synchrony,Regularity,Locality,Pipelinability*. These computation arrays are very useful since most scientific and signal processing computations can be represented in Signal Flow Graphs, and most of these graphs can be systematically converted into Systolic Arrays [3].

To indicate the breadth of the systolic approach, the following is a partial list of problems for which systolic solutions exist [2]:

Signal and Image processing:

- FIR and IIR filtering and 1-D convolution;
- 2-D convolution and correlation;
- discrete Fourier transformation;
- interpolation;
- 1-D and 2-D median filtering; and

Matrix arithmetic:

- matrix-vector multiplication;
- matrix-matrix multiplication;
- matrix triangularization;
- QR-decomposition;
- solution to Toeplitz linear systems;
- singular value decomposition; and
- eigenvalue problems.

Added to the above is a whole list of non-numeric applications. Many alternatives exist for the implementation of systolic algorithms at both chip and board levels. Let us list various kinds just to have an idea about the flexibility involved [2]:

1. Single-purpose systolic array: It is a special purpose systolic array processor built just for that specific Algorithm.

2. Multi-purpose systolic array: The approach in designing such arrays is based on the observation that many systolic algorithms can be executed on systolic arrays of very similar structures.

3. Non-programmable building-block: This consists of building block

processors capable of executing a few predefined, and commonly used functions.

4. Programmable systolic arrays: These are systolic array processors, where a fixed number of programmable PE's are connected together in a certain manner, possibly with other control circuits.

The exploitation of parallelism in an algorithm has many steps and should take into account many time and space constraints [5]:

a- Each algorithm could be represented by a dependence graph which generates dependency constraints.

b- Use same processors for different calculations but with one schedul. This should help minimizing the space complexity but puts constraints on the scheduling process.

c- The communication between different processors has to be accounted for.

d- Use of *Single Assignment Language* as a way of taking advantage of parallelism in an algorithm.

We have seen the advantages in designing Optimal Systolic Arrays for various steps and parts of the Conjugate Gradient method. The adoption of the pipeline architecture, which puts these steps together to represent the whole algorithm, becomes very practical with minimum Time and Space complexities.

3. Design of Optimal Arrays for Matrix Arithmetic

Different approaches will be taken toward designing optimal arrays which represent various matrix arithmetic. We will be looking into the operations which are repeatedly used in many steps of the Conjugate Gradient algorithm. Let us define some parameters which decide the general structure of our processor [4].

Parameter 1 - Velocity of Data Flow: The velocity of a datum x is defined as the directional distance passed by x during a clock cycle and is denoted by $\vec{x_d}$.

Parameter 2 - Data Distribution: For a two-dimensional array X used as input or output of a systolic array , the elements along a row or a column are arranged in a straight line and are equally spaced as they pass through the systolic array, and the relative positions of the elements are iteration independent. The *row displacement* of X is defined as the directional distance between $x_{i,j}$ and $x_{i+1,j}$ as X passes throught the systolic array and is denoted by

$\vec{x}_{is}$.

Similarly, the *column displacement of X* ($\vec{x}_{js}$) *is defined as a directional distance between* $x_{i,j}$ *and* $x_{i,j+1}$.

Parameter 3 - Period: Suppose the time at which a computation is performed is defined by the function τ_c, and the time at which an input is accessed for a particular computation is τ_a. The *periods of i and j* for two-dimensional outputs are defined as

$$t_i=\tau_c(z^k_{i+1,j})-\tau_c(z^k_{i,j}) \tag{3.1}$$

$$t_j=\tau_c(z^k_{i,j+1})-\tau_c(z^k_{i,j}) \tag{3.2}$$

In computing $z^k_{i,j}=f[z^{k-1}_{i,j},x(i,k),y(k,j)]$, it is assumed that the recurrence is expressed in a backward form or has been converted into a backward form. Define the *period of iterative computation* for two-dimensional outputs as

$$t_k=\tau_c(z^{k+1}_{i,j})-\tau_c(z^k_{i,j}) \tag{3.3}$$

Note that τ_k is always positive. Define the *periods of X and Y with respect to k* in the computation of $z_{i,j}$ as the time between accessing successive elements of X and Y. Formally

$$t_{kx}=\tau_a(x_{i,k+1})-\tau_a(x_{i,k}) \tag{3.4}$$

$$t_{ky}=\tau_a(y_{k+1,j})-\tau_a(y_{k,j}) \tag{3.5}$$

t_{kx} and t_{ky} may be negative depending on the order of access defined in the subscript-access functions x(i,k) and y(k,j). Since data needed in the computation of $z^{k+1}_{i,j}$ after the computation of $z^k_{i,j}$ must be assembled in time t_k, it is true that

$$t_k=|t_{kx}|=|t_{ky}|.$$

There is a total of thirteen parameters for two-dimensional linear recurrences, of which three are for the velocities of data flow, $\vec{x}_d$, $\vec{y}_d$, $\vec{z}_d$, six are for data distributions, $\vec{x}_{is}$, $\vec{x}_{js}$, $\vec{y}_{is}$, $\vec{y}_{js}$, $\vec{z}_{is}$, $\vec{z}_{js}$, and four are for the periods, t_{kx}, t_{ky}, t_i, t_j. These parameters can be used in constraint equations to govern the correctness of the design and in performance measures to define the number

of PE's needed and the completion time. The following theorem states the relationships among these parameters.

Theorem 1 (Theorem of Systolic Processing) [4]: Suppose a two-dimensional recurrence computation $z_{i,j}^{k} = f[z_{i,j}^{k-1}, x(i,k), y(k,j)]$ is implemented in a systolic array, then the velocities, data distributions, and periods must satisfy the following vector equations:

$$t_{kx}\vec{x_d}+\vec{x_{ks}}=t_{kx}\vec{z_d} \tag{3.6}$$

$$t_{ky}\vec{y_d}+\vec{y_{ks}}=t_{ky}\vec{z_d} \tag{3.7}$$

$$t_i\vec{x_d}+\vec{x_{is}}=t_i\vec{y_d} \tag{3.8}$$

$$t_i\vec{z_d}+\vec{z_{is}}=t_i\vec{y_d} \tag{3.9}$$

$$t_j\vec{y_d}+\vec{y_{js}}=t_j\vec{x_d} \tag{3.10}$$

$$t_j\vec{z_d}+\vec{z_{js}}=t_j\vec{x_d} \tag{3.11}$$

3.1 Matrix - Matrix Multiplication: Let us now apply the above definitions and theorem on the following recurrence equation which represent the matrix multiplication of X (n×n) by Y (n×n) to give Z (n×n).

$$z_{i,j}^{0} = 0$$

$$z_{i,j}^{k} = z_{i,j}^{k-1} + x_{i,k}y_{k,j}$$

These equations are true for $1 \le i, j, k \le n$. The design problem is formulated as:

minimize T subject to (3.6) to (3.11) and

$$\frac{1}{t_{jmax}} \le |\vec{x_d}| \le 1 \text{ or } |\vec{x_d}| = 0 \tag{3.12}$$

$$\frac{1}{t_{imax}} \le |\vec{y_d}| \le 1 \text{ or } |\vec{y_d}| = 0 \tag{3.13}$$

$$\frac{1}{t_{kmax}} \le |\vec{z_d}| \le 1 \text{ or } |\vec{z_d}| = 0 \tag{3.14}$$

$$1 \le |t_k| \le t_{kmax};\ 1 \le |t_i| \le t_{imax};\ 1 \le |t_j| \le t_{jmax} \tag{3.15}$$

$$|t_k|\,|\vec{z_d}| = k_1 \le t_{kmax};\ |t_i|\,|\vec{y_d}| = k_2 \le t_{imax};\ |t_j|\,|\vec{x_d}| = k_3 \le t_{jmax} \tag{3.16}$$

$$|\vec{x_{is}}| \neq 0 \; ; |\vec{x_{ks}}| \neq 0 \; ; |\vec{y_{ks}}| \neq 0 \tag{3.17}$$

$$|\vec{y_{js}}| \neq 0 \; ; |\vec{z_{is}}| \neq 0 \; ; |\vec{z_{js}}| \neq 0 \tag{3.18}$$

In our example (X×Y = Z) these matrices are flowing in three different directions (see Fig.1) and have (2n - 1) streams of data flow, hence the #PE for X and Y is $(2n-1)^2$. However, matrix Z is flowing in a different direction which cuts off two corners in the PE configuration. #PE is reduced by 2n. The time needed for multiplying two n-by-n matrices is $nt_k+(n-1)|t_i|+(n-1)|t_j|$ since it takes nt_k steps to compute $z_{1,1}$, $(n-1)|t_i|$ steps from computing $z_{1,1}$ to $z_{n,1}$, and $(n-1)\ |t_j|$ steps from computing $z_{1,n}$ to $z_{n,n}$. In the design (Fig.9-1), no load or drain time is necessary. In order for systolic processing to be more efficient than a serial computation, it is necessary for $T \leq T_{serial}$. By using the minimum values for two of the periods ($t_k=1$, $|t_j|=1$, $|t_i|=1$) in inequalities (3.12) to (3.18) , the upper bound for the other period (t_{kmax}, t_{imax}, t_{jmax}) is obtained.

It is noted that systolic designs for linear recurrence equations are independent of the problem size. To reduce the search complexity, an optimal design for a smaller problem can be found (Fig. 10-1 for n=3) and is used to extend to the systolic design for a larger version of the same problem [4].

To minimize the completion time, the different directions of data flows are first determined. The maximum values of t_k, t_i, and t_j are found. The speeds of the data flows are evaluated from (3.16) by using $k_1 = k_2 = k_3 = 1$. The six remaining unknowns on spatial distributions can be solved from the systolic processing equations (3.6) to (3.11).

By using the minimum values for $t_k=t_i=t_j=1$ we found a feasible solution which satisfies the constraint equations (3.6) to (3.18) and the resulting completion time T(n) and the #PE are given by:

$$T(n) = 3n - 2$$

$$\#PE = 4n^2 - 6n + 1$$

3.2 Matrix - Vector Multiplication: Similar approach is taken toward the design of Optimal Arrays to represent the Matrix Vector Multiplication. A×X = Y where A is (m×n) matrix, X is (n×1) vector, and Y is (m×1) vector.

This multiplication is represented by the following recurrence equation:

$$y_i^0 = 0$$

$$y_i^k = y_i^{k-1} + a_{ik} x_k$$

For i=1,.....,m and k=1,.....n

With an objective function of T(n)×PE, we take advantage of the *Complete Binary Tree* which could be built for the addition of elements. If n is not a power of two, we add more zeros to make it one, otherwise n is suitable for our structure. The following theorem is of great importance to us in deriving the equations for the completion time T(n) and the #PE.

Theorem 2 (Theorem of Complete Binary Trees) [1]: If v is the number of leaves in a complete binary tree, then the height of that tree is $h = \log_2 v$, and the total number of nodes in the same tree is # nodes $= 2^{h+1} - 1$.

Using the above theorem, and the architecture presented in (Fig.10-2) we can conclude that:

$$T(n) = m + \log_2 n$$

$$\#PE = 2n - 1$$

Since we have one tree which has (2n - 1) PE's.

3.3 Vector - Vector Multiplication: In Vector-Vector multiplication, if T(n) is the objective function then one Complete Binary Tree is needed and is presented in (Fig.10-3) with the following results:

$$T(n) = 1 + \log_2 n$$

$$\#PE = 2n - 1$$

3.4 Vector - Vector Addition/Substraction: In Vector Vector addition/substraction, the objective function is T(n), the corresponding array is shown in (Fig.10-4) with the following results:

$$T(n) = 1$$

$$\#PE = n$$

Where n is the size of the vectors (X, Y,and V) in

$$X +/- Y = V.$$

4. Summary and results

We can list the different speed-up and processor utilization of each processor machine we analysed, with the following definitions [4]:

$$\text{Processor–Utilization} = \frac{\text{Speed–Up}}{\#PE} \text{ where}$$

$$\text{Speed–Up} = \frac{T_{best-sequential}}{T_{parallel}}$$

i)- In $Matrix_{n\times n} \times Matrix_{n\times n}$ we have [1]:

$$S\text{–}U = \frac{n^{2.81}+18(\frac{n}{2})^2}{3n-2}$$

$$P\text{–}U = \frac{n^{2.81}+18(\frac{n}{2})^2}{(3n-2)(4n^2-6n+1)}$$

ii)- In $Matrix_{m\times n} \times Vector_{n\times 1}$ we have :

$$S\text{–}U = \frac{2n\times m}{m+\log_2 n}$$

$$P\text{–}U = \frac{2n\times m}{(m+\log_2 n)(2n-1)}$$

iii)- In $Vector_{1\times n} \times Vector_{n\times 1}$ we have :

$$S\text{–}U = \frac{2n}{1+\log_2 n}$$

$$P\text{–}U = \frac{2n}{(1+\log_2 n)(2n-1)}$$

Knowing that the best one can hope to achieve is a speed-up equal to #PE and a processor utilization of one. Let us try to get a better understanding of what our design achieved by considering a numerical example.

Type of Oper.	Size	S-U	P-U
2 $Matrices_{nxn}$	n=300	47K	0.13
$Mat._{mxn} x Vect._{nx1}$	$m=10^3$ $n=10^4$	20K	0.602
$(Vect. x Vect.)_n$	$n=10^4$	1.33K	0.05

From the above table we see the *large S-U (Speed-Up)* , and the *small P-U (Processor Utilization) except* in Mat.×Vect. , that is caused by the nature of the respective objective functions which were minimized in our design (i.e. T(n) in first and third operations, and #PE × T(n) in the second operation).

5. Conclusion

After designing these arrays, trees, and vectors in different architectures, the question of the realization of such parallel machines is raised. Available VLSI chips have a maximum number of transistors that can be built in one chip. This number is in the neighborhood of 500K. In our case all of the PE's are either scalar adders, or scalar multipliers, with the worst case (in arbitrary structure of square matrix multiplication) being $2(4n^2-6n+1)$. In other words the maximum value that n can take is $\max(n) \approx 333$ (in using one systolic array only). Therefore for bigger dimensions, partitioning of the given matrix is considered. On the other hand, for other parallel machines designed for Matrix × Vector, and Vector × Vector ,$\max(n) \approx 250K$ which is very encouraging. Another subject could be raised about what kind of accuracy the used scalar adders and scalar multipliers have. In this area we can achieve 32 bit floating point using serial bit operations, and if highly parallel architecture is requested, this could be another independent project and design by itself.

We have seen the numerous advantages in designing Optimal Systolic Arrays for various steps and parts of the Conjugate Gradient method. The adoption of the pipeline architecture, which puts these steps together to represent the whole algorithm, becomes very practical with minimum Time and Space complexities.

6. Bibliography

[1] Aho, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Comp., 1974, pp. 115-232.

[2] H.T. Kung, *Systolic Algorithms*, Acedemic Press Inc., 1984, pp. 127-137.

[3] Sun-Yuan Kung, *On Supercomputing with Systolic/Wavefront Arrays Processors*, IEEE proc., vol. 72, NO. 7, July 1984, pp. 867-883.

[4] Guo-Jie Li and Benjamin W. Wah, *The Design of Optimal Systolic Arrays*, IEEE Trans. Comput., vol. c-34, Jan 1985, pp. 66-77.

[5] W. L. Miranker and A. Winkler, *Spacetime Representations of Computational Structures*, Computing, vol. 32, 1984, pp. 93-114.

[6] Harold A. Sabbagh, Computing External Fields due to a Whip Source and Foil Target, 1988, pp 1-33.

[7] Harold Siegel and L. Jamieson, *Algorithmically Specified Parallel Computers*, May 1984, pp. 160-170.

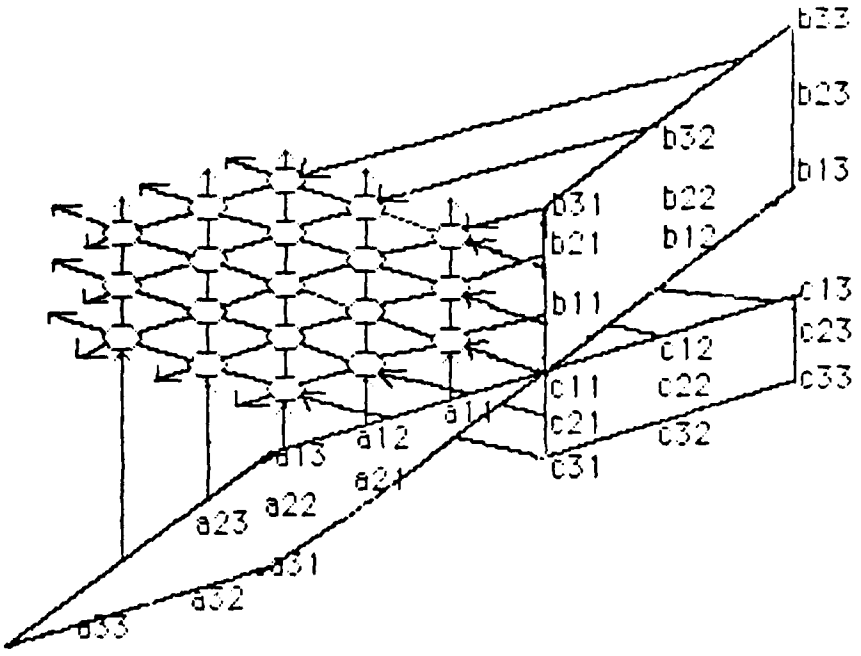
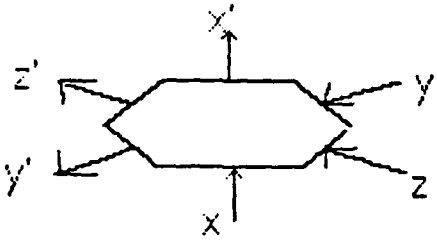


FIGURE 10 - 1: SYSTOLIC ARRAY FOR A x B = C
in three dimensions.
T(n) = 7 clock periods
# of PE's = 19; i.e. 19 scalar adders,
19 scalar multipliers

Individual cell: (has one scalar adder, and one scalar multiplier)



Input: x,y,z
Output : x' = x
y' = y
z' = z + xy

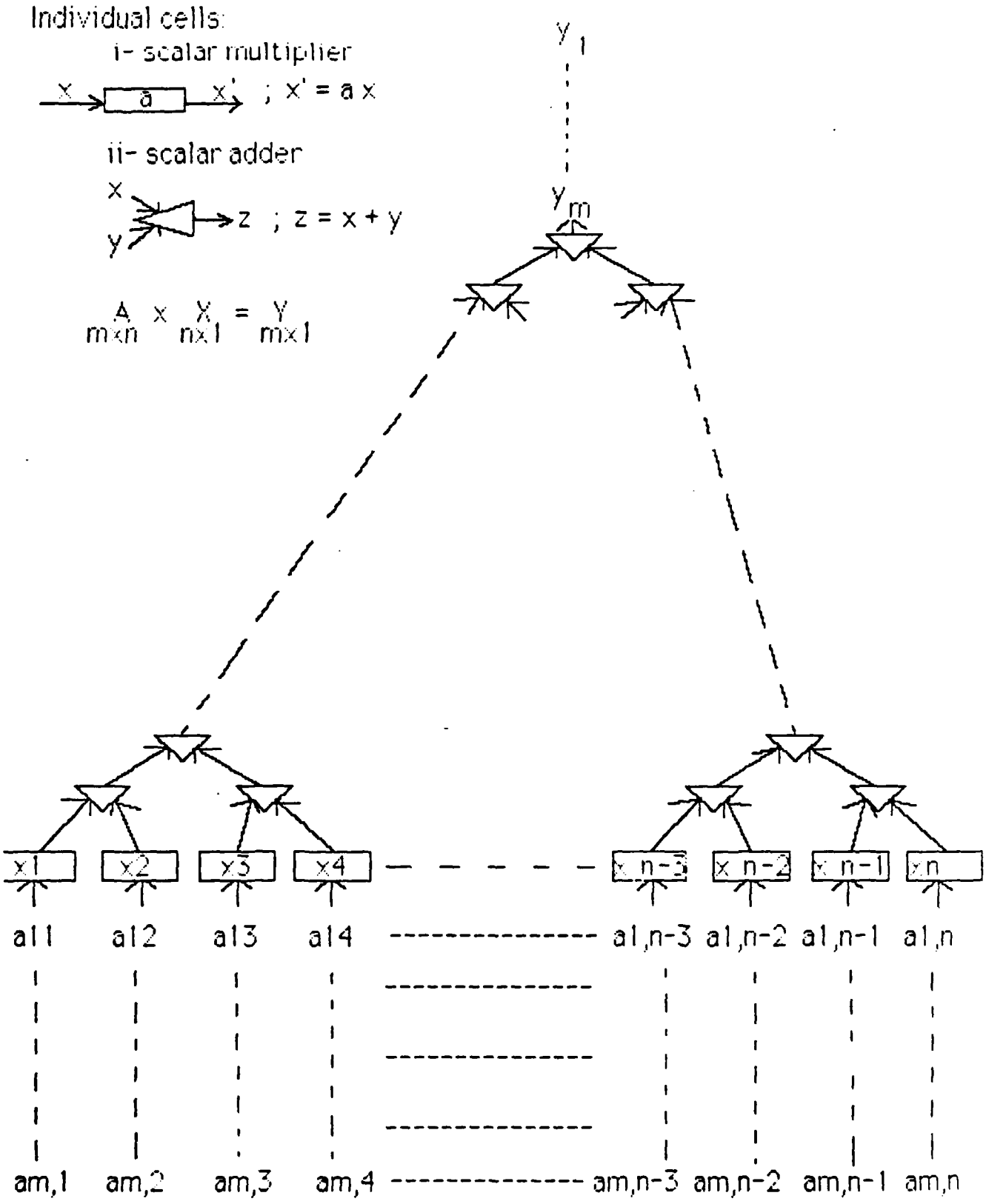


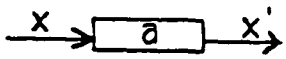
FIGURE 10-2 : MATRIX x VECTOR

$T = m + \log_2 n$ clock periods

# of PE'S = 2n - 1
or n scalar multipliers, and n-1 scalar adders

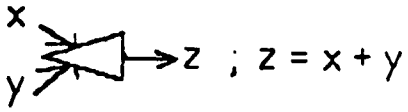
Individual cells:

i- scalar multiplier



; x' = a x

ii- scalar adder



; z = x + y

$$V_{1\times n} \times X_{n\times 1} = Y_{1\times 1}$$

FIGURE 10-3 : VECTOR x VECTOR

$T = 1 + \log_2 n$ clock periods

# of PE'S = 2n - 1
or n scalar multipliers, and n-1 scalar adders

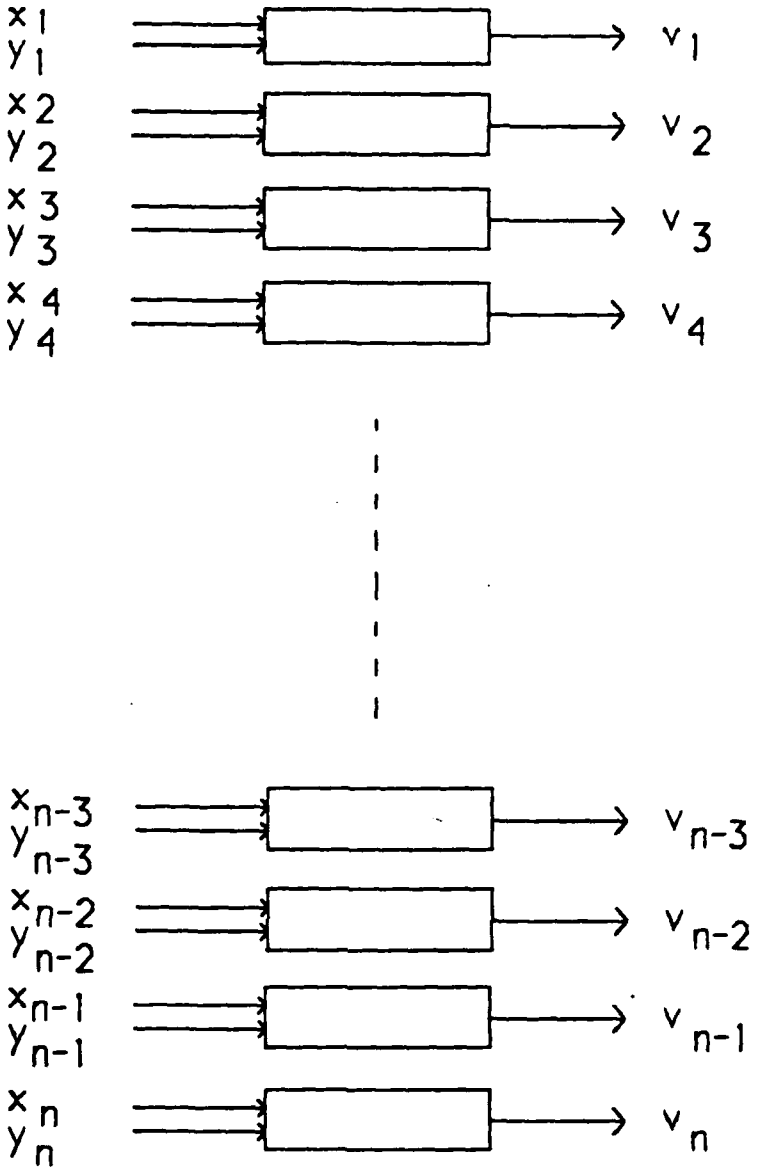


FIGURE 10-4 : VECTOR +/- VECTOR
T(n) = 1 clock periods; # of PE's = n scalar adders
X +/- Y = V (vectors of size n)

Individual cell:

Input : x,y
Output : v = x +/- y

CHAPTER XI

QUALITATIVE ANALYSIS OF ELECTROMAGNETIC FIELDS FOR FLAW DETECTION

QUALITATIVE ANALYSIS OF ELECTROMAGNETIC FIELDS FOR FLAW DETECTION

1.0 Overview

As an alternative to the quantitative methods typically used for non-destructive evaluation of materials here at Sabbagh Associates, the possibility of using qualitative methods to detect and classify flaws was explored. The tact used was to examine electromagnetic fields for features which would yield information as to the presence and nature of flaws. We use the term "artificial intelligence" to describe the techniques tried since the programs developed mimic the way a human being might detect and classify a flaw by examining a graph of a magnetic field.

Models for examing EMFs of two types of materials were developed, on for stainless steel and one for graphite composite. The stainless steel version was developed more extensively, since accurate computer models for depicting an EMF of a field for a specified flaw exist. These models, developed at Sabbagh Associates, allow the user to define the size, shape, and location of a flaw in a material, as well as the frequency the data was gathered at and the nature of the material used. The techniques developed using the computer model were also test on laboratory data, with favorable results.

Only lab data was used in developing the program for graphite composite since accurate models for graphite composite have not yet been developed. As a result the effect of flaw size and depth on the EMF could not be tested fully, since the physical materials can not be easily manipulated to test for different flaws. However, the groundwork is laid for further refinement once the computer models for graphite composite materials are developed.

2.0 Qualitatively Analyzing the Data

2.1 Patterns in the Data

After examing z, phi, and radial data, radial data was chose as the field type to use because the signals for radial data obtained during actual collection are stronger than for z or phi data since the sensors can be placed closer to the tube.

2.2 The Flaw Series

A set of 16 flaw series was developed to determine how the shape of the graph of the EMF was affected by the nature of the flaw. Notice that the dimensions increase by a factor of 2. These flaws were designed to give an easy reference to the effect of the flaw dimensions on the field perturbation; i.e. doubling the width of flaw T increases the height of the peak by so much; etc. Each of the flaws in the series came in nine different depths: 2/10, 4/10, 6/10, 8/10 on the inside of the tube, all the way through, and 8/10, 6/10, 4/10, 2/10 of the way through on the outside. The flaw series is listed in the table below:

		Length			
		8	16	32	64
Width	10	tT	tS	nS	vS
	20	tR	sT	S	lS
	40	sR	R	T	lT
	80	vR	wR	wT	bT

First, to see how EMFs changed with flaw depth and frequency, the graphs of each were generated and printed (see figures 1a-e). Several discoveries were made. First, it was determined that the dimensions of the swellings were essentially the same in the z and phi directions for different frequencies and depths, varying only in height. Since the width and length of the swellings remained fairly constant over frequency and flaw depth, it was deemed likely that they could give an accurate indication of the length and width of the flaw. For example, flaw R4 has a length of 16 z-units, and 40 phi-units. Every eighth point was plotted. By examining the graph "eyeballometrically," we can see the distance between the two peaks is approximately 2, which is 1/8th the flaw length. Likewise, if we measure the distance at half the bulge's height, we see it is somewhat greater than 4, close to 5, which is approximately 1/8 of the flaw's width. This holds true for the other flaws in the R series, as well as the flaws in the S and T series. Therefore measuring the peak-to-peak distance and the width at half-height will give a good estimate of the flaw shape in the z and phi directions.

By examining the maximum values for each plot over a range of frequencies, we noticed that the values followed a pattern. Therefore the data was examined again and plotted in a two-dimensional graph according to the maximum values over the range 10kHz to 5MHz. It was found that we got the maximum response for each flaw in the real data at around 37 kHz, no matter what the dimensions of the flaw in the z, phi, and r directions were—a phenomenon which no one here was able to explain. Since the data gathered at Purdue didn't use frequencies that low, we haven't been able to verify this experimentally. It does, however, suggest the concept of a representative frequency which would yield a value

indicative of the flaw depth. Also, we noticed that when the flaw was on the outside of the tube the response dropped off in the higher frequencies. This suggests that measuring at a low frequency and again at a high frequency and comparing the results will indicate wheather the flaw is on the inside or the outside of the tube, since the value will drop off significantly when the flaw is on the outside. (See figures 2a-d.)

2.3 Organizing Data by Depth

To show the effect of the flaw depth on the EMF, the data was plotted according to flaw depth. To further show the effect of changing the width and length of the flaw, flaws with similar widths but varying lengths were plotted together, and flaws with similar lengths but varying widths were plotted together. These plots were made at three different frequencies: 200kHz, 700kHz, and 4MHz. It was found that increasing both the width and the length of the flaw causes a greater peak height, but increasing the width causes a slightly greater increase than increasing the length.

It was hoped that some pattern would emerge clearly showing the depth of the flaw by looking at the height of the peak on the graph. Unfortunately the results were not as promising as expected, as the values for all flaw depths on the inside and all flaw depths on the outside were too close together to be distinguished clearly, particularly when considering the presence of noise. The tests did prove promising in one respect, however. The response for flaws on the inside of the tube was significantly greater than flaws on the outside (as in figures 2a-d), so if nothing else we could determine which side of the tube the flaw was on by examining the peak height using this method.

3.0 The Stainless Steel Model

3.1 Flaw Length and Width Estimation

While different flaws produce "bumps" in a graph of differing dimensions, the graph itself always has the same basic form: two bumps aligned along the z-axis. It was suggested in earlier quarterly reports that by examining these bumps a fairly accurate estimation of the flaw could be made. A C program was developed to test and refine this idea. The results were promising.

It was thought an accurate estimate of the flaw length could be obtained by measuring

the distance between the peaks, and an estimate of the flaw width by measuring the width of a bump at half-height. So, the field was read in to a two-dimensional array and a qualitative analysis performed to obtain these measurements. So that the accuracy of the estimation could be judged, a facility for entering the flaw dimensions was added. Two versions of the program were implemented: one for reading data from the Sabbagh Model, and one for reading lab data gathered at Purdue from a previous project. A discussion of the program and test results follows. The program was first developed for stainless steel materials. Two versions were developed, one for analyzing data from the Sabbagh Model and one for analyzing lab data. This was because the field dimensions and usefulness of the data differed between the two.

3.2 Reading Data.

Data is "gathered" from binary data files residing on the Alliant in such a way as to simulate a two-pass collection of data with a sensor. At first, every eighth point along the *z-* and *phi*-axes is read in and stored in the array. This constitutes the initial pass. Then this coarse-resolution field is examined and the location of the flaw is found. Then a second pass is made, centering around the region within the side and end points that are less than 1/8 the peak value. Every point of data is read this time, giving fine resolution (see figures 3a-b).

3.3 Estimating Flaw Length and Width.

When the flaw has been located, the distance between the peaks along the *z*-axis is measured, giving an estimation of the flaw length. The width of each of the two bumps at half-height is also measured, giving an estimation of the width at each end. The actual dimensions of the flaw can then be entered and an idea of the accuracy can be gotten either graphically or verbally (see figures 4a-8f).

Real, imaginary, and magnitude data was tested to see which gave the most accurate representation of the flaw. Imaginary data proved to be inaccurate, often overestimating the flaw size by a considerable amount. Real and magnitude data were both more accurate than imaginary data, but the real data occasionally would become irregular, such as might happen when the flaw was on the outside of the tube (see NAVAIR II quarterly report #1). Magnitude data proved both the most accurate and the most consistent, so it was chosen as the standard form. (Phase data was ignored, since it gives no useful information with this algorithm.)

We also wanted to overify that the algorithm would work for lab data as well as model data. 500kHz data was chosen for length and width estimation both because it seemed to provide useful information, and because we have lab data from Purdue at this frequency.

The field can also be measured at 4MHz to determine whether the field is on the inside or the outside.

3.4 Performance.

The algorithm was tested with all the flaws in the flaw series. The results were promising. When the flaw was of sufficient size, the algorithm gave very accurate results (see figures 5a-b). When the flaw was smaller in either the *z* or *phi* direction, the algorithm tended to exaggerate the size of the flaw.

The algorithm was also tested with data for flaws that were available both in the Purdue data and the model data. The algorithm gave similar results for each (see figures 6a-7f). This demonstrates the feasability of this algorithm for use on real data, as well as the accuracy of the Sabbagh model.

When the flaw runs all the way around the circumference of the tube, the bumps go all the way around as well instead of dying off (see figures 8a-f). In this case the original algorithm won't work, since you can't measure the width at half height of the bump. So, the ability to distinguish between slot-type flaws (those discussed previously) and uniform thinning flaws was added (see figures 8a-c). This suggests the possibility of "tuning" the algorithm to look for and distinguish between the fields of different flaw types. If a certain type of flaw were expected in a material, the data could be interpreted in such a way as to represent a flaw of the expected type. If we knew, for example, that flaws were likely to be long but very thin (resembling flaw h in the Purdue data), we could use the distance between the peaks as the length estimate, but make the width estimate much narrower or ignore it altogether.

One problem with the algorithm in its current state is that it considers all flaws to have a regular rectangular shape. Where a flaw is of a different shape error could result. This occurred in the test for flaw e, which is deeper in the middle than it is on the ends. The algorithm underestimated the length. As mentioned before, the algorithm could be adjusted to take this into account if it were likely that non-rectangular flaws would be found. Where the expected flaw type varies a lot this could be a problem, since it would make it difficult to predict the type of flaw represented by the bumps in the field. However, it is likely that a flaw of similar size to the actual flaw would be predicted, which would still have some value.

Where greater accuracy is required than could currently be provided by this algorithm, we could still use it to locate flaws and define a "region of containment" within which we know the flaw would be found. We could then apply a quantitative algorithm such as that discribed by Sabbagh & Sabbagh (Final Report, DOD #DE-AC02-83ER80096, and *Review of Progress in Quantitative Nondestructive Evaluation*, Vol 6A, Thompson & Chimenti, ed.), which would give greater accuracy but which is considerably more time consuming than the method described here. By defining a region of containment, we could

have the quantitative algorithm only look at parts of the field that are useful, ignoring more irrelevant regions.

3.1 Depth Estimation by Indexing and Interpolation

Recall that when the field was calculated for frequencies ranging from 10kHz to 5MHz and the magnitude peak values measured and plotted, there was consistently a maximum response at about 40kHz. This phenomenon occurred regardless of flaw size and depth (see 2nd Quarterly Report, NAVAIR II). Upon re-examining this data, it was noticed that the peak value increased regularly with flaw depth. The data was replotted at this frequency for all sixteen flaws in the series according to depth. Sure enough, for a given flaw length and width, the peak value increased noticably as the depth increased. This was true for flaws on both the inside and outside of the tube. A scale could be constructed for known depth values, and then when a flaw of unknown depth is found we could measure the peak value and interpolate between known values, giving an estimation of the depth of the new flaw. In addition to flaw depth, the peak values also increased with flaw width and length. therefore our scale had to be three-dimensional, so that the effect of the length and width could be taken into account as well as the depth.

The peak values increased with flaw size at a different rate depending on whether the flaw was on the inside or the outide. It was also noticed that when the field was measured at a high frequency (4MHz), peak values were at a consistent level when the flaw was on the inside, but dropped near zero when the flaw was on the outside. This was due to the shallow field depth at high frequencies. This fortunately provides us with a method of determining whether the flaw is on the inside or the outside. Two three-dimensional tables of known peak values at 40kHz can be constructed, one for the inside and one for the outside, and by measuring the peak value at 4MHz we can determine which table to use to estimate the flaw depth.

3.5 Implementation

A C program was written incorporating two tables containing peak values for all the flaws in the series at seven depths each (0, 1/10, 2/10, 4/10, 6/10, 8/10, and 1 times the thickness of tube). When running the program, the user inputs the flaw length and width (assuming the length and width can be determined using other means, as in section 2), and the maximum values at 40kHz and 4MHz. The program then decides whether to use the inside or outside flaw table by comparing the 4MHz value with a threshold (0.3 was chosen since it lay inbetween the measured values for the inside and outside peak values). Then the length, width, and 40kHz value are interpolated between values found in the appropriate three-dimensional table, resulting in an interpolated value for the depth.

3.3 Results

To test the program, data was generated using the Sabbagh model for various flaws at 40kHz and 4MHz and the maximum value for each field was taken. Peak values for flaws used to create the interpolation tables were of course right on. For flaws with the same widths and depths as those used to create the tables, but differing in depth, the results were less accurate. For flaws having widths an depths similar to those used to create the table, the results were off by around 2%. Where the length, width and depth all differred from table values, the error was higher (about 10%) due to the multiple interpolations. Data for flaws on the outside of the tube (farther from the sensor) are always less accurate than for flaws on the inside, and this instance is no exception, as error was typically about 33%. Once, a moderately deep outside flaw was mistaken for an inside flaw, since the peak value for the flaw at 4MHz was greater than the inside/outside threshold value. (Obviously, the algorithm needs a little tweaking.)

Unfortunately this algorithm is not currently testable on lab data, since we don't have any data taken at 40kHz. (Experts suggest that the sensors used to gather data are not appropriate for collection at frequencies below 100kHz, although a different sensor design might give good results.) Also, it is likely that results would be misleading when used on non-rectangular flaws. However, several ideas were learned in this program: 1) that it is possible to find a frequency which will give the best results for a given classification scheme, and 2) by taking data at a high frequency, it is possible to determine whether a flaw is on the inside or the outside of the material.

4.0 The Graphite Composite Model

After the stainless steel program was developed, the techniques developed for identifying flaws in stainless steel were adapted for graphite composite materials. Since the model for graphite composite EMF's is not fully developed yet, the program was designed to use actual data gathered in the lab. Currently, the program can depict and manipulate EMF's either on the terminal screen, on hard copy, or on a high-resolution graphics monitor.

4.1 Description of Program

Some of the lab data had a slope to it, so that values on one end were significantly greater than values on the other end. In order to better discern aberrations due to flaws, the data was sent through a filter which calculated a best-fit plane with the data and subtracted the planar values to level the data. This filtered data was then read in by the program and organized to allow detailed analysis of the field.

As with the stainless steel version, either the real, imaginary, or magitude data of the

EMF can be viewed. When real and imaginary data are seen in grayscale representation, flaw bumps appear like actual depressions in the material due to an illusion (see figure 11). For manipulation and analysis, magnitude data proved to be more useful since all the bumps have positive values and are more easily distinguishable from the background, although the grayscale image was not as appealing (see figure 12).

The program was developed using C on the Alliant. To take advantage of the high-resolution graphics monitor, a version was also made which ran on the IBM AT. For convenience, the field can be printed out on the screen using digits from 0 to 9 representing low to high points in the field, a feature available on both versions of the program. On the AT, output can be toggled from the terminal to the graphics monitor.

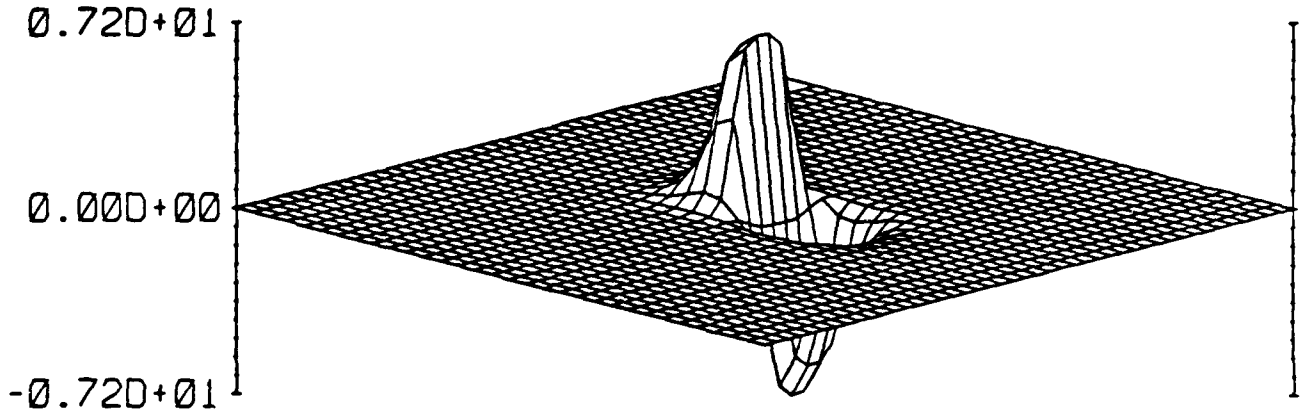
To clarify the location of flaws, the flaw field can be weighted in such a way as to distinguish the flaw from the background. Two methods are implemented: a masking technique which allows you to incrementally increase the contrast between high and low points, and a flaw region locator which will pick out and highlight regions where a flaw is likely (see figure 13). Each method has its advantages: with masking the image can be tuned to pick out the most likely flaw locations. The region locator can also pick out likely flaw locations, but without the fine-tunability. It can weed out individual high points that are likely to have been caused by noise and do not indicate a flaw.

4.2 Suggestions for Future Development

What we have developed here so far is a methodology for detecting flaws in graphite composites. This methodology could be improved by making the program smarter, able to do things automatically that currently require operator intervention. The flaw region discernment feature could be tuned to take into account the magnitude of individual points compared with the surrounding points to more intelligently decide whether a high point value is due to noise or the presence of a flaw. The weighting of the masking feature could also be made more intelligent, so that EMFs with loud background would use a different weighting scheme than quieter ones to identify the flaws more accurately. The functionality of the masking and region locator techniques could be combined for a more cohesive flaw detection scheme.

Thus far we have concentrated on flaw detection only. No attempt has been made to determine the size and depth of the actual flaw. When the graphite composite model has been refined further, we will be able to use it to better understand the relationship between flaw size and depth and the shape of the EMF.

R4
200



	RANGE	STEP	
x-axis:	1- 32	1.0	fcnmin = -0.72449D+01
y-axis:	1- 45	1.0	fcnmax = 0.68689D+01

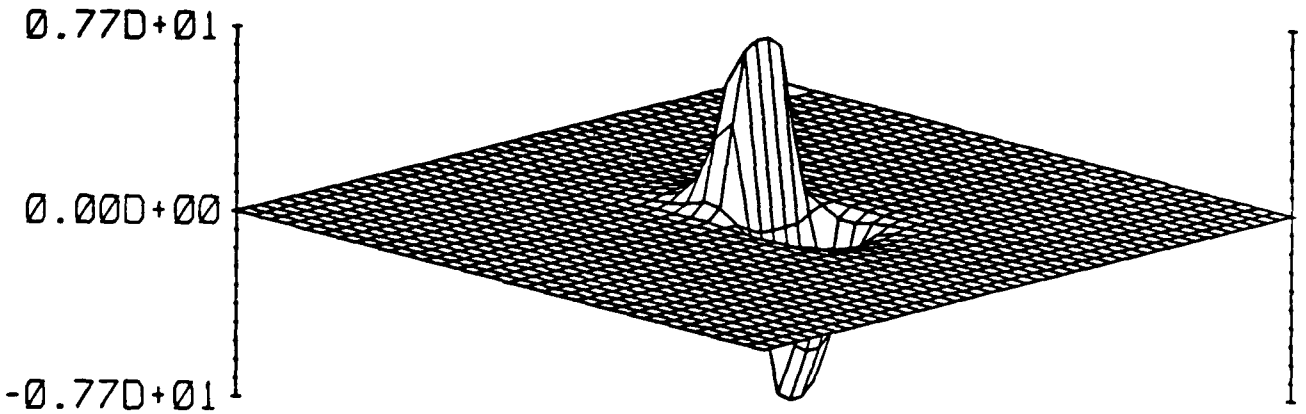
WED, MAR 25 1987 10:53:32

CREATED BY PROGRAM GRFPAK›APP›PLOT3D›PLOT3D

Figure 1A

R4 at 200 KHZ
REAL DATA

R4
400



	RANGE	STEP	
x-axis:	1- 32	1.0	fcnmin = -0.77327D+01
y-axis:	1- 45	1.0	fcnmax = 0.73979D+01

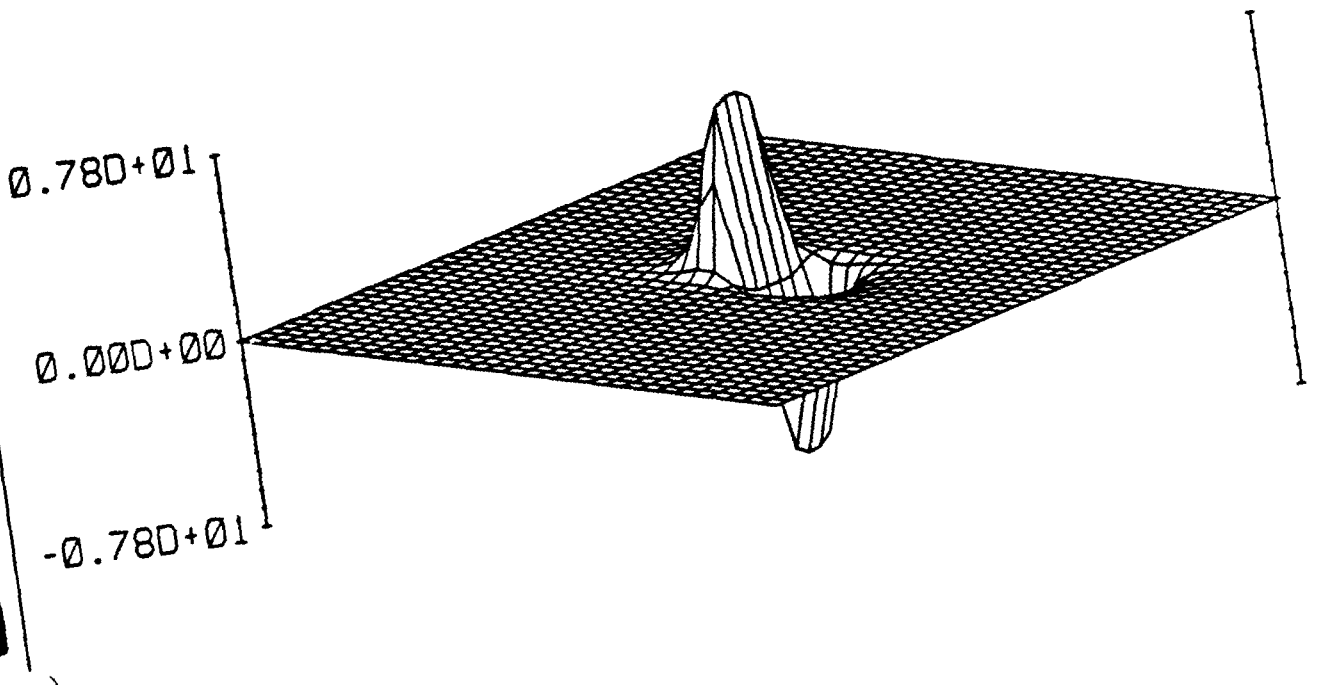
WED, MAR 25 1987 10:49:19

CREATED BY PROGRAM GRFPAK›APP›PLOT3D›PLOT3D

Figure 4B

R4 at 400 KHZ
REAL DATA

R4
600



x-axis: RANGE 1- 32 STEP 1.0
y-axis: 1- 45 1.0

fcnmin = -0.78297D+01
fcnmax = 0.75335D+01

10:43:15

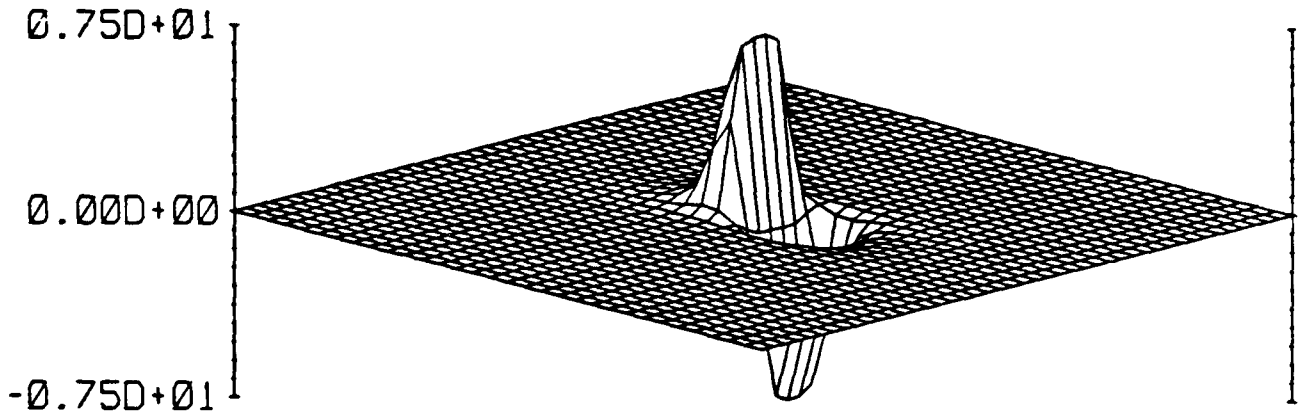
WED, MAR 25 1987

CREATED BY PROGRAM GRFPAK›APP›PLOT3D›PLOT3D

Figure 1c

R4 at 600 KHZ
REAL DATA

R4 1
800



	RANGE	STEP	
x-axis:	1- 32	1.0	fcnmin = -0.75209D+01
y-axis:	1- 45	1.0	fcnmax = 0.72648D+01

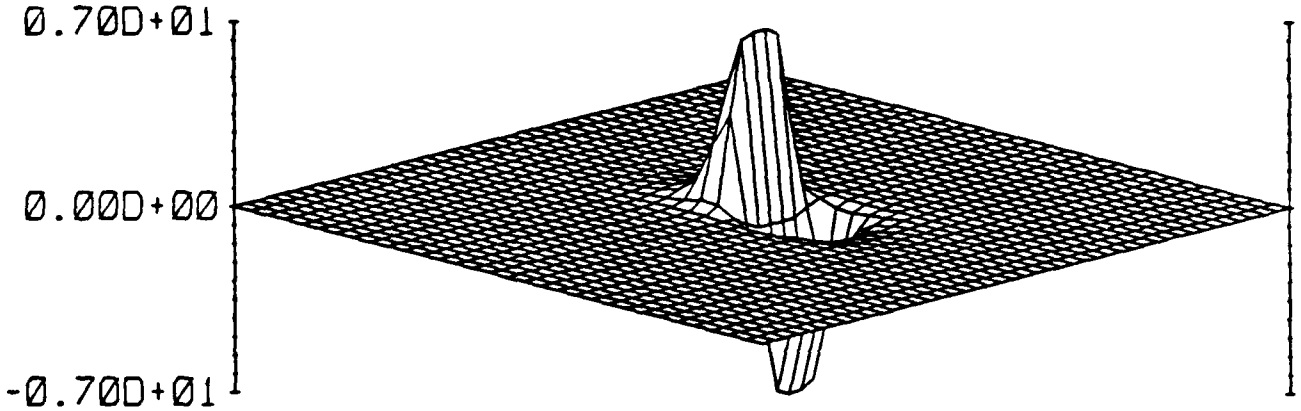
WED, MAR 25 1987 10:38:28

CREATED BY PROGRAM GRFPAK›APP›PLOT3D›PLOT3D

Figure 1D

R4 at 800 KHZ
REAL DATA

R4
1000



	RANGE	STEP	
x-axis:	1- 32	1.0	fcnmin = -0.70462D+01
y-axis:	1- 45	1.0	fcnmax = 0.68258D+01

WED, MAR 25 1987 10:34:50

CREATED BY PROGRAM GRFPAK›APP›PLOT3D›PLOT3D

Figure 2E

R4 at 1MHZ
REAL DATA

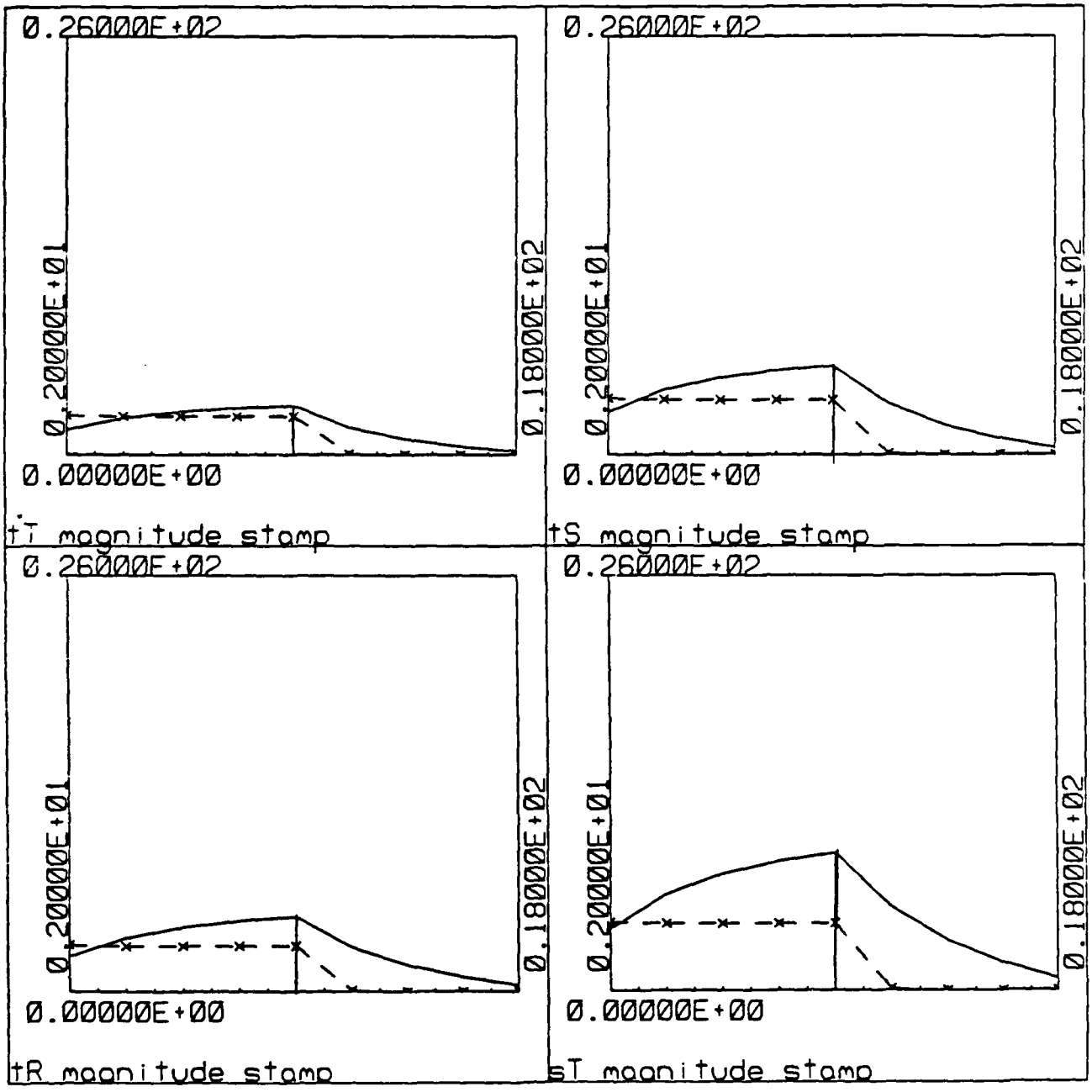


Figure 2a

Figures 2a-d: plots of the flaws in the flaw series according to depth. Solid lines are at 40kHz; dashed lines are at 4MHz. the line in the center shows the peak value when the flaw goes all the way through; it gets progressively shallower towards the edges. Inside flaws are on the left, outside flaws are on the right.

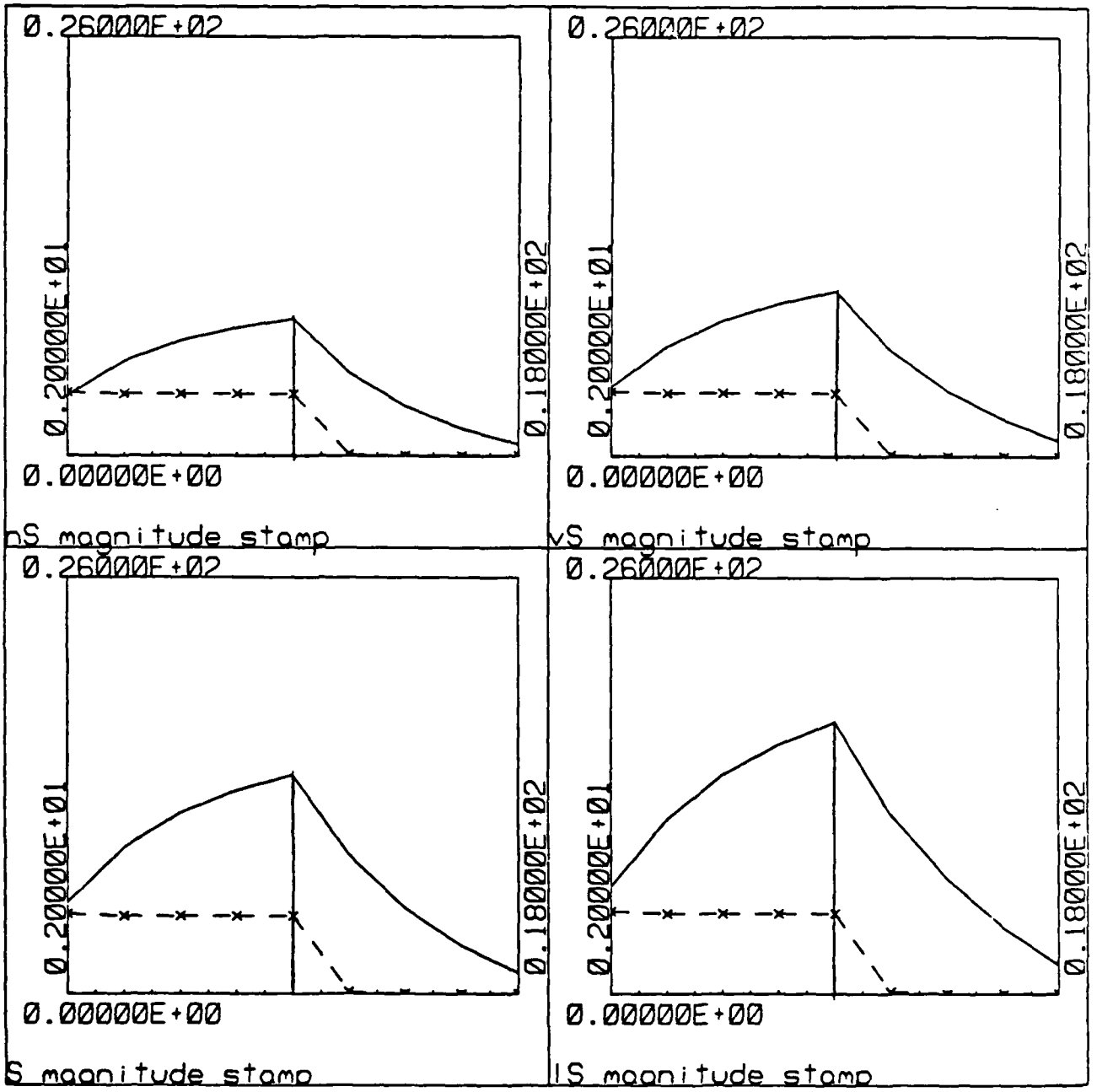

0.26000E+02
0.20000E+01
0.18000E+02
0.00000E+00
nS magnitude stamp
0.26000E+02
0.20000E+01
0.18000E+02
0.00000E+00
vS magnitude stamp
0.26000E+02
0.20000E+01
0.18000E+02
0.00000E+00
S magnitude stamp
0.26000E+02
0.20000E+01
0.18000E+02
0.00000E+00
lS magnitude stamp

Figure 1b

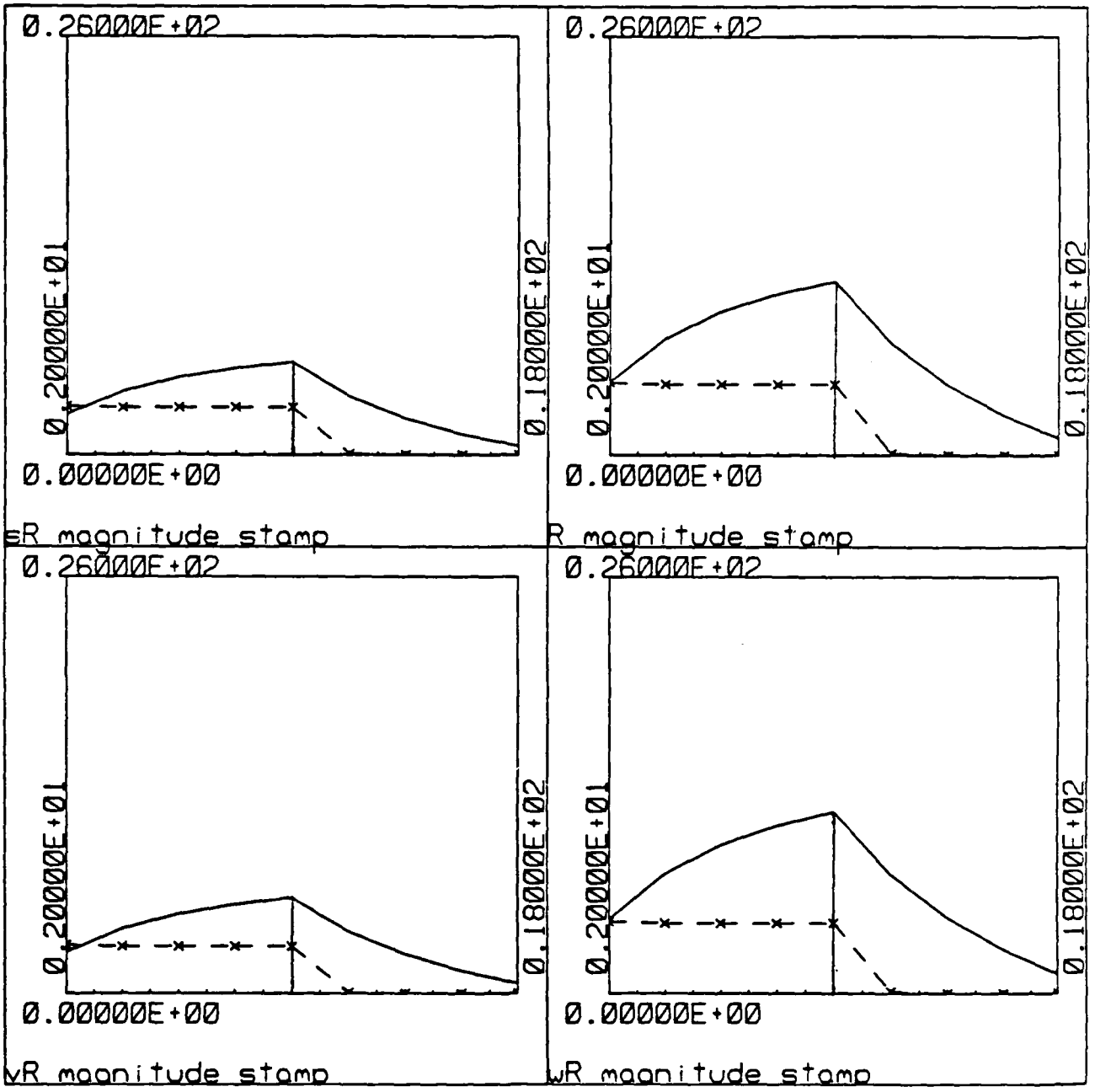

0.26000E+02
0.20000E+01
0.18000E+02
0.00000E+00
R magnitude stamp
0.26000E+02
0.20000E+01
0.18000E+02
0.00000E+00
R magnitude stamp
0.26000E+02
0.20000E+01
0.18000E+02
0.00000E+00
R magnitude stamp
0.26000E+02
0.20000E+01
0.18000E+02
0.00000E+00
R magnitude stamp

Figure 2c

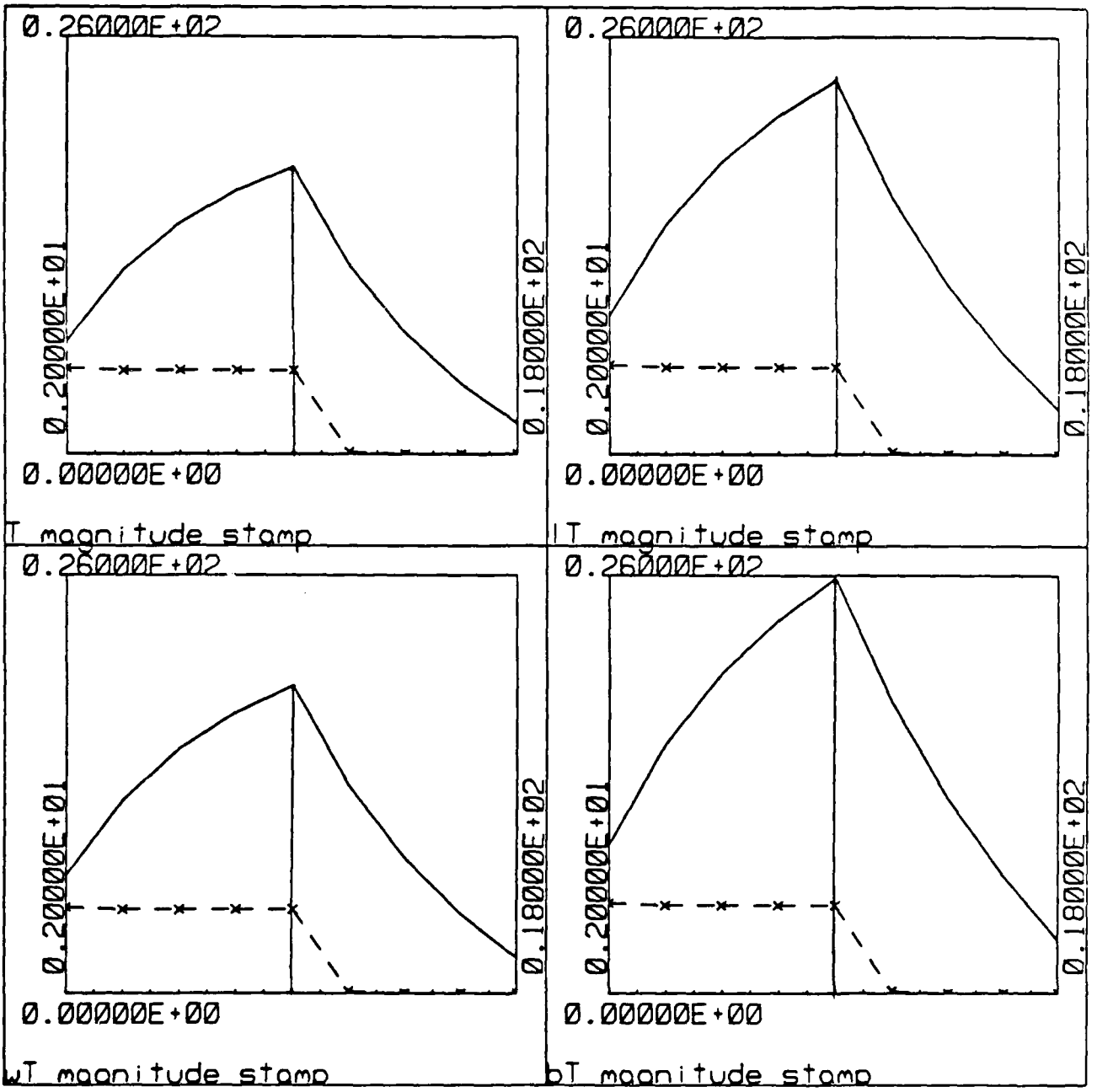
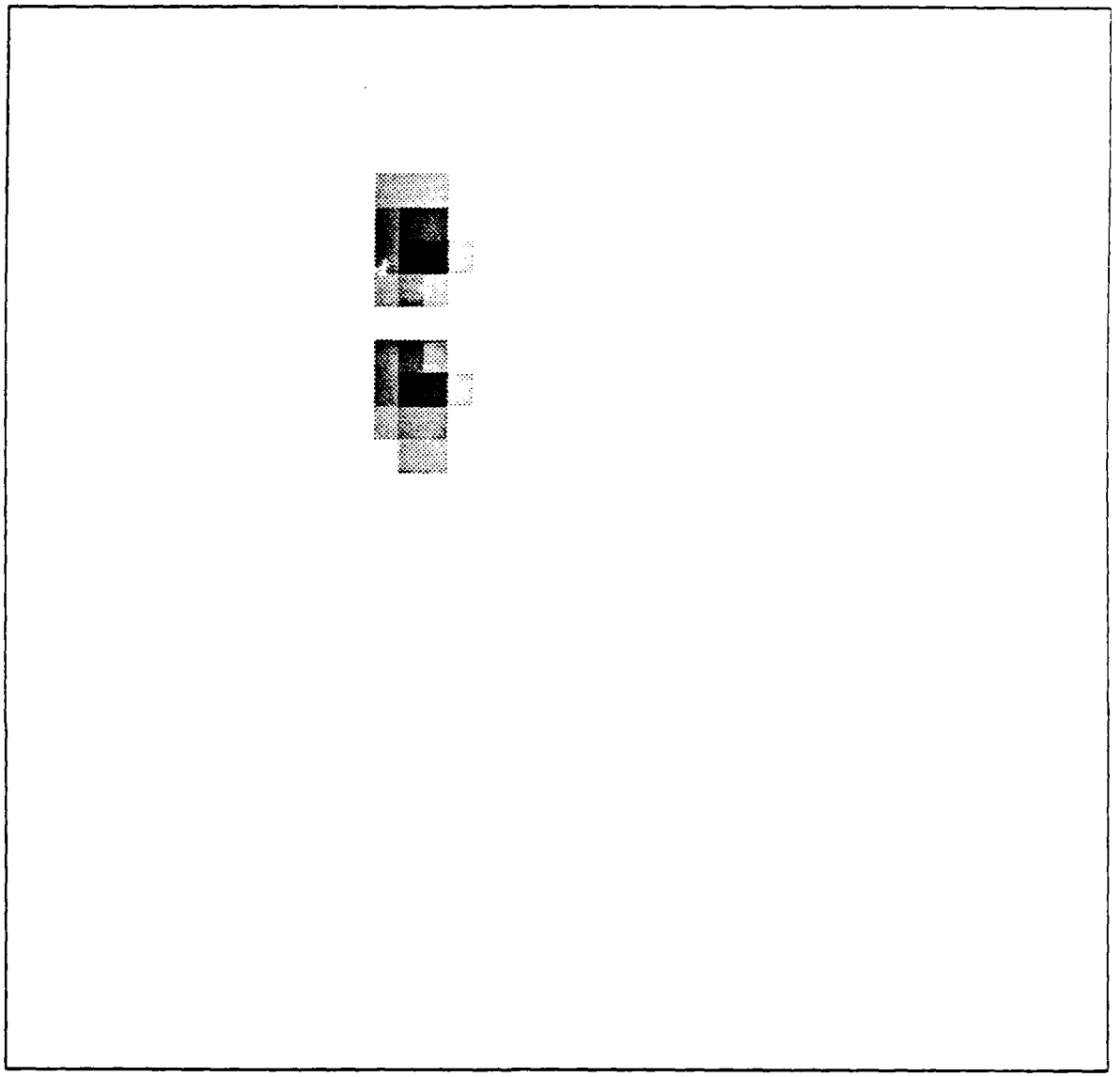

0.26000E+02
0.20000E+01
0.18000E+02
0.00000E+00
T magnitude stamp
0.26000E+02
0.20000E+01
0.18000E+02
0.00000E+00
IT magnitude stamp
0.26000E+02
0.20000E+01
0.18000E+02
0.00000E+00
wT magnitude stamp
0.26000E+02
0.20000E+01
0.18000E+02
0.00000E+00
bT magnitude stamp

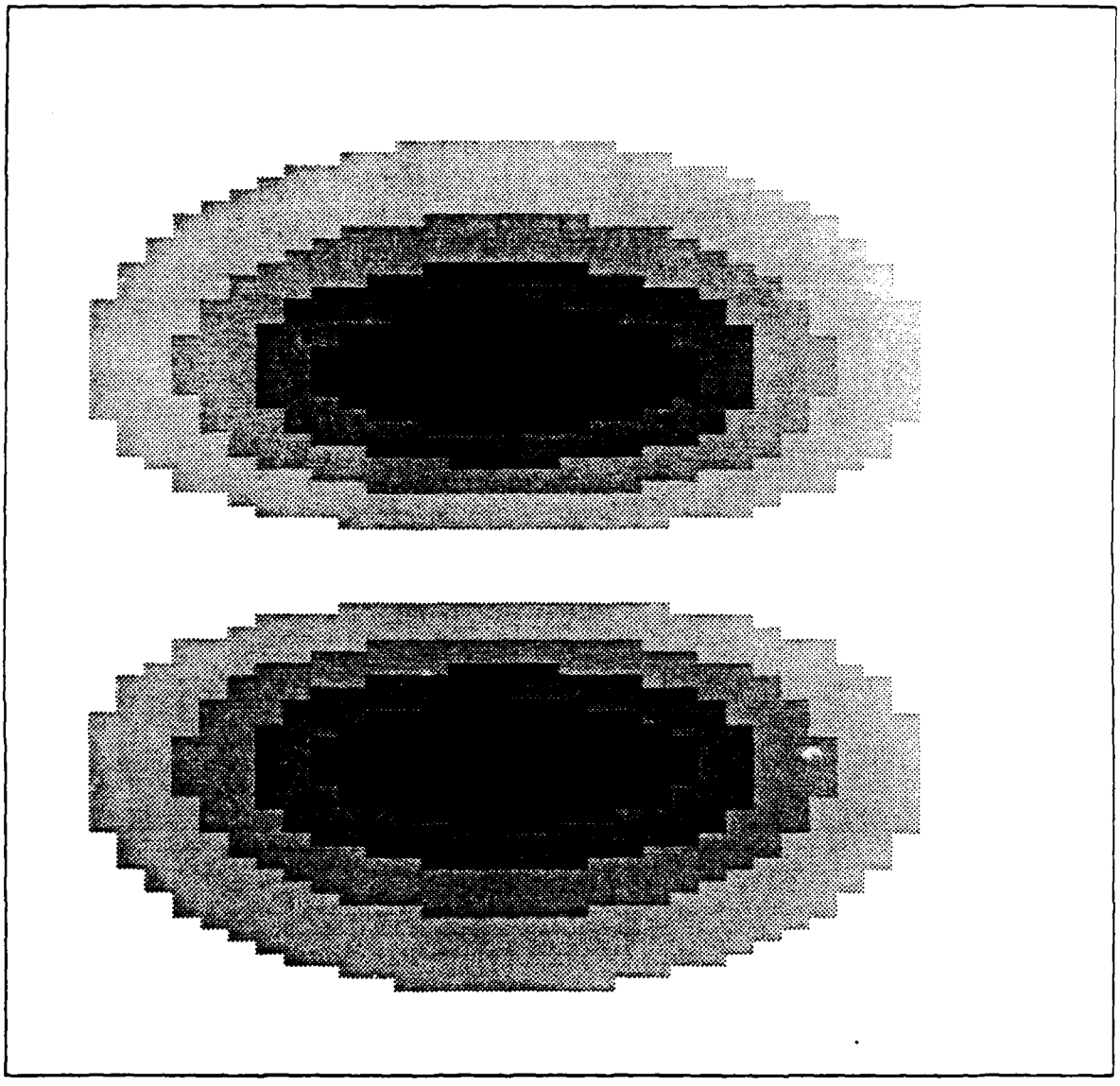
Figure 3d



Flaw mS61

Figure 3a

Figures 3a & 3b illustrate the zeroing-in mechanism. On the first pass a flaw is found in a section of tube 1 inch long by locating bumps in the magnetic field (figure 3a). On the second pass, the field is measured around the flaw in much finer resolution (figure 3b).



Flaw mS62

Figure 3b

```
using flaw S2

enter command> r
reading data from file /c/navdat2/cache/hr_fS2.500kHz:
    zstart = 0, phistart = 0, skip = 8...
they say data was read in okay.

enter command> t
set to read at zstart = 88, phistart = 160, zlength = 80, philength = 40.

enter command> r
reading data from file /c/navdat2/cache/hr_fS2.500kHz:
    zstart = 88, phistart = 160, skip = 1...
they say data was read in okay.

enter command> a
reading data from file /c/navdat2/cache/hr_fS2.4MHz, zstart = 88, phistart = 160, skip = 1...
they say data was read in okay.

enter command> gc
enter the corners as they were originally defined> -9 10 -15 16

enter command> p
flaw identified.

enter command> s

enter command> sp
data type: m.
maximum value = 9.161450, [z,phi] = [144,179]
4MHz max = 4.968279, flaw is on the inside.
corner[0] = 5.429874 @ [112,170]
corner[1] = 5.429874 @ [112,189]
corner[2] = 5.429874 @ [143,170]
corner[3] = 5.429874 @ [143,189]
flaw S2 is within region [111,169], [111,190], [144,169], [144,190]
estimated flaw length = 0.14 inches;  estimated flaw width = 0.15 inches.
actual flaw length = 0.13 inches;  actual flaw width = 0.14 inches.
absolute length error = 0.01 inches;  absolute width error = 0.01 inches.
relative length error = %6.25;  relative width error = %10.00.

enter command> q
```

Figure 4a

Figures 4a-2d: program runs for flaws S2 (0.0096 inches deep, inside), S6 (0.0288 inches deep, inside), S-4 (0.0288 in deep, outside), S-8 (0.0096 in. deep, outside). Note error measurements.

```
using flaw S6

enter command> r
reading data from file /c/navdat2/cache/hr_fS6.500kHz:
    zstart = 0, phistart = 0, skip = 8...
they say data was read in okay.

enter command> t
set to read at zstart = 88, phistart = 160, zlength = 80, philength = 40.

enter command> r
reading data from file /c/navdat2/cache/hr_fS6.500kHz:
    zstart = 88, phistart = 160, skip = 1...
they say data was read in okay.

enter command> a
reading data from file /c/navdat2/cache/hr_fS6.4MHz, zstart = 88, phistart = 160, skip = 1...
they say data was read in okay.

enter command> gc
enter the corners as they were originally defined> -9 10 -15 16

enter command> p
flaw identified.

enter command> sp
data type: m.
maximum value = 9.866004, [z,phi] = [144,179]
flaw is on the inside.
corner[0] = 5.828905 @ [112,170]
corner[1] = 5.828905 @ [112,189]
corner[2] = 5.828905 @ [143,170]
corner[3] = 5.828905 @ [143,189]
flaw S6 is within region [111,169], [111,190], [144,169], [144,190]
estimated flaw length = 0.14 inches;  estimated flaw width = 0.15 inches.
actual flaw length = 0.13 inches;  actual flaw width = 0.14 inches.
absolute length error = 0.01 inches;  absolute width error = 0.01 inches.
relative length error = %6.25;  relative width error = %10.00.

enter command> q
```

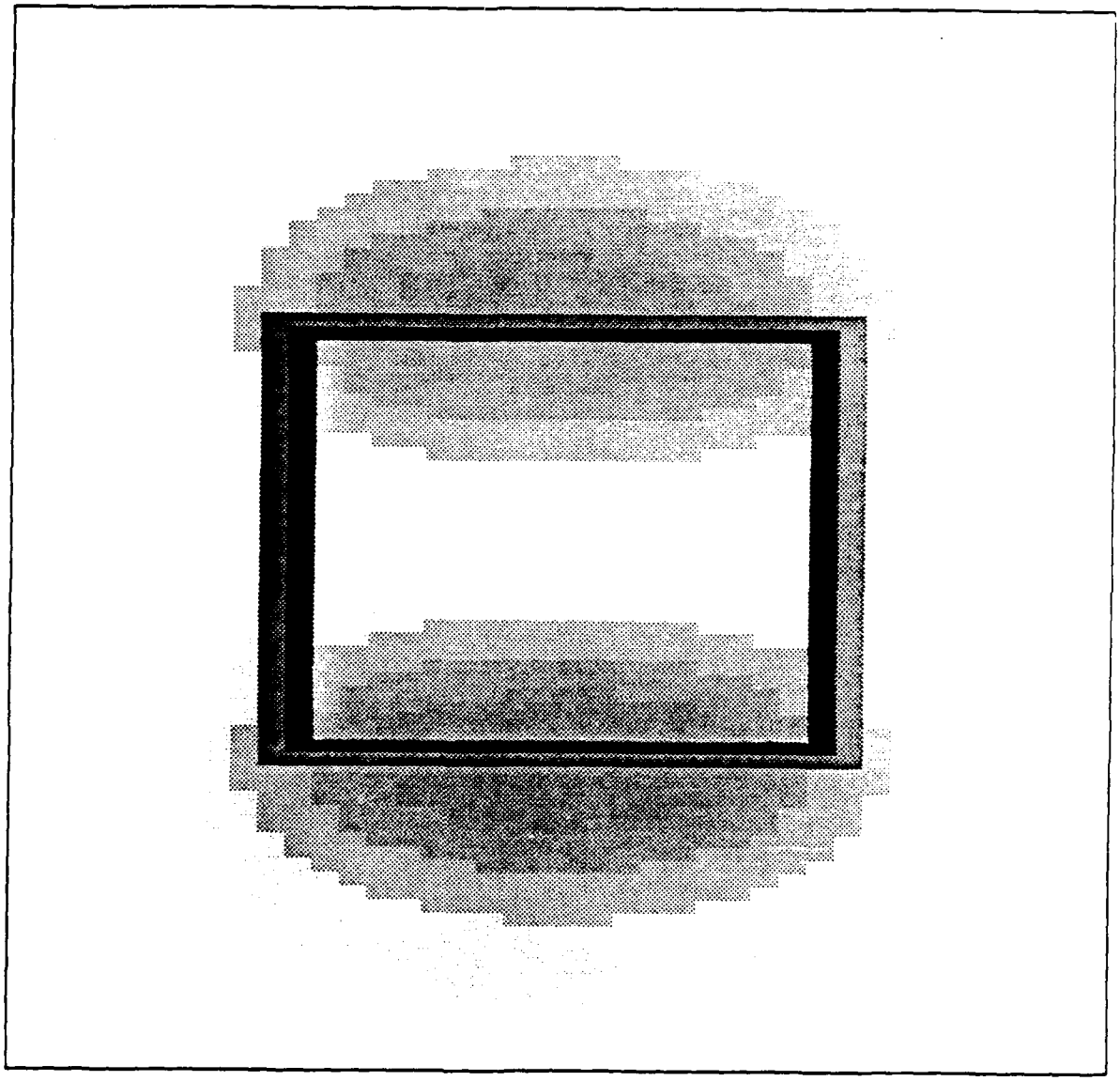
Figure 4b

```
using flaw S-4

enter command> r
reading data from file /c/navdat2/cache/hr_fS-4.500kHz:
    zstart = 0, phistart = 0, skip = 8...
they say data was read in okay.

enter command> t
set to read at zstart = 80, phistart = 160, zlength = 96, philength = 40.

enter command> r
reading data from file /c/navdat2/cache/hr_fS-4.500kHz:
    zstart = 80, phistart = 160, skip = 1...
they say data was read in okay.

enter command> a
reading data from file /c/navdat2/cache/hr_fS-4.4MHz, zstart = 80, phistart = 160, skip = 1...
they say data was read in okay.

enter command> gc
enter the corners as they were originally defined> -9 10 -15 16

enter command> p
flaw identified.

enter command> sp
data type: m.
maximum value = 0.536622, [z,phi] = [144,179]
flaw is on the inside.
corner[0] = 0.319718 @ [112,170]
corner[1] = 0.319718 @ [112,189]
corner[2] = 0.319718 @ [143,170]
corner[3] = 0.319718 @ [143,189]
flaw S-4 is within region [111,169], [111,190], [144,169], [144,190]
estimated flaw length = 0.14 inches;  estimated flaw width = 0.15 inches.
actual flaw length = 0.13 inches;  actual flaw width = 0.14 inches.
absolute length error = 0.01 inches;  absolute width error = 0.01 inches.
relative length error = %6.25;  relative width error = %10.00.

enter command> q
```

Figure 4c

```
using flaw S-8

enter command> r
reading data from file /c/navdat2/cache/hr_fS-8.500kHz:
    zstart = 0, phistart = 0, skip = 8...
they say data was read in okay.

enter command> t
set to read at zstart = 72, phistart = 152, zlength = 112, philength = 56.

enter command> r
reading data from file /c/navdat2/cache/hr_fS-8.500kHz:
    zstart = 72, phistart = 152, skip = 1...
they say data was read in okay.

enter command> a
reading data from file /c/navdat2/cache/hr_fS-8.4MHz, zstart = 72, phistart = 152, skip = 1...
they say data was read in okay.

enter command> gc
enter the corners as they were originally defined> -9 10 -15 16

enter command> p
flaw identified.

enter command> s

enter command> sp
data type: m.
maximum value = 0.071131, [z,phi] = [110,179]
4MHz max = 0.000002, flaw is on the outside.
corner[0] = 0.044968 @ [112,170]
corner[1] = 0.044968 @ [112,189]
corner[2] = 0.044968 @ [143,170]
corner[3] = 0.044968 @ [143,189]
flaw S-8 is within region [110,168], [110,191], [145,168], [145,191]
estimated flaw length = 0.14 inches;  estimated flaw width = 0.16 inches.
actual flaw length = 0.13 inches;  actual flaw width = 0.14 inches.
absolute length error = 0.02 inches;  absolute width error = 0.03 inches.
relative length error = %12.50;  relative width error = %20.00.

enter command> q
```

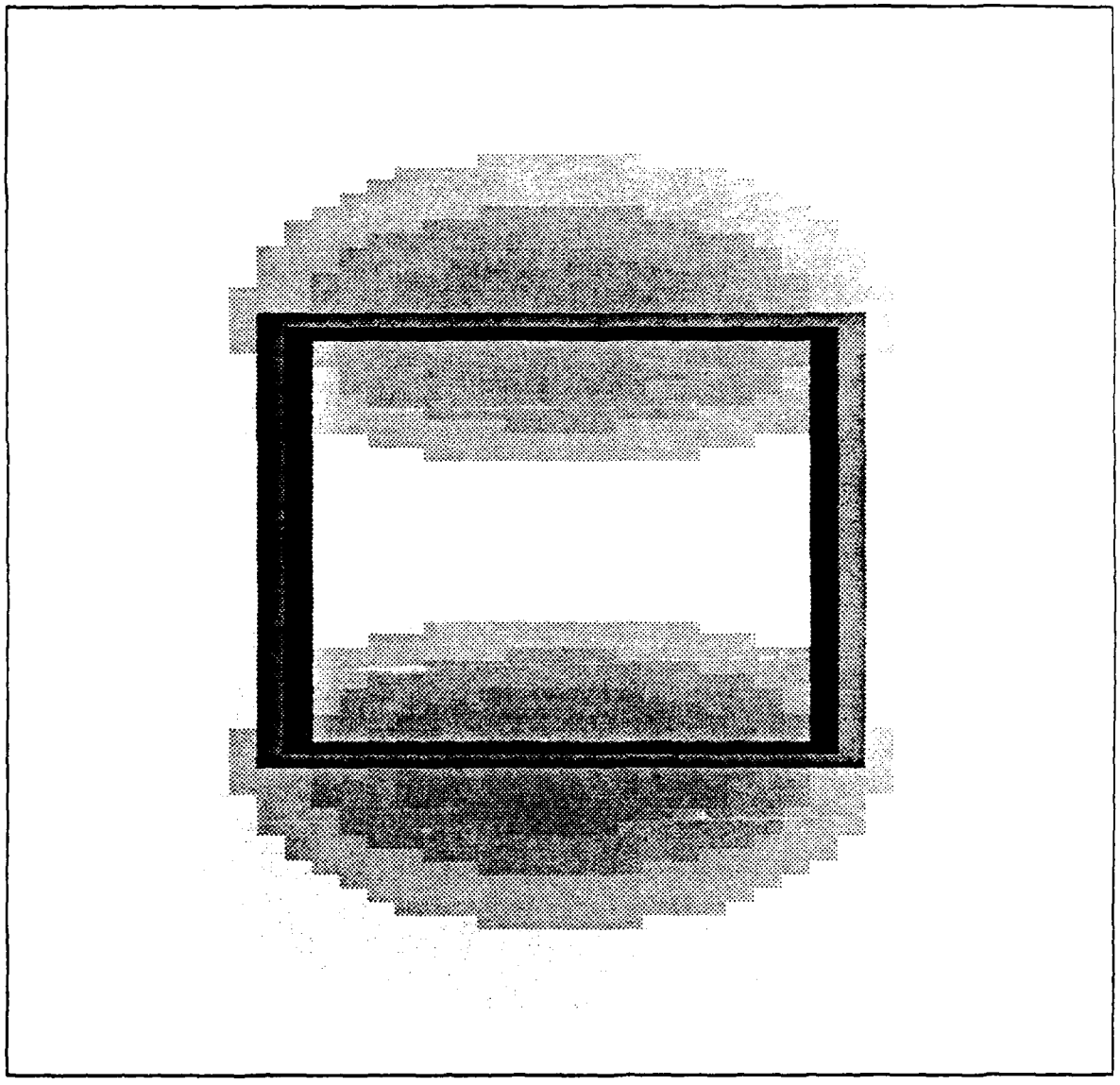
Figure 4d



Flaw S2

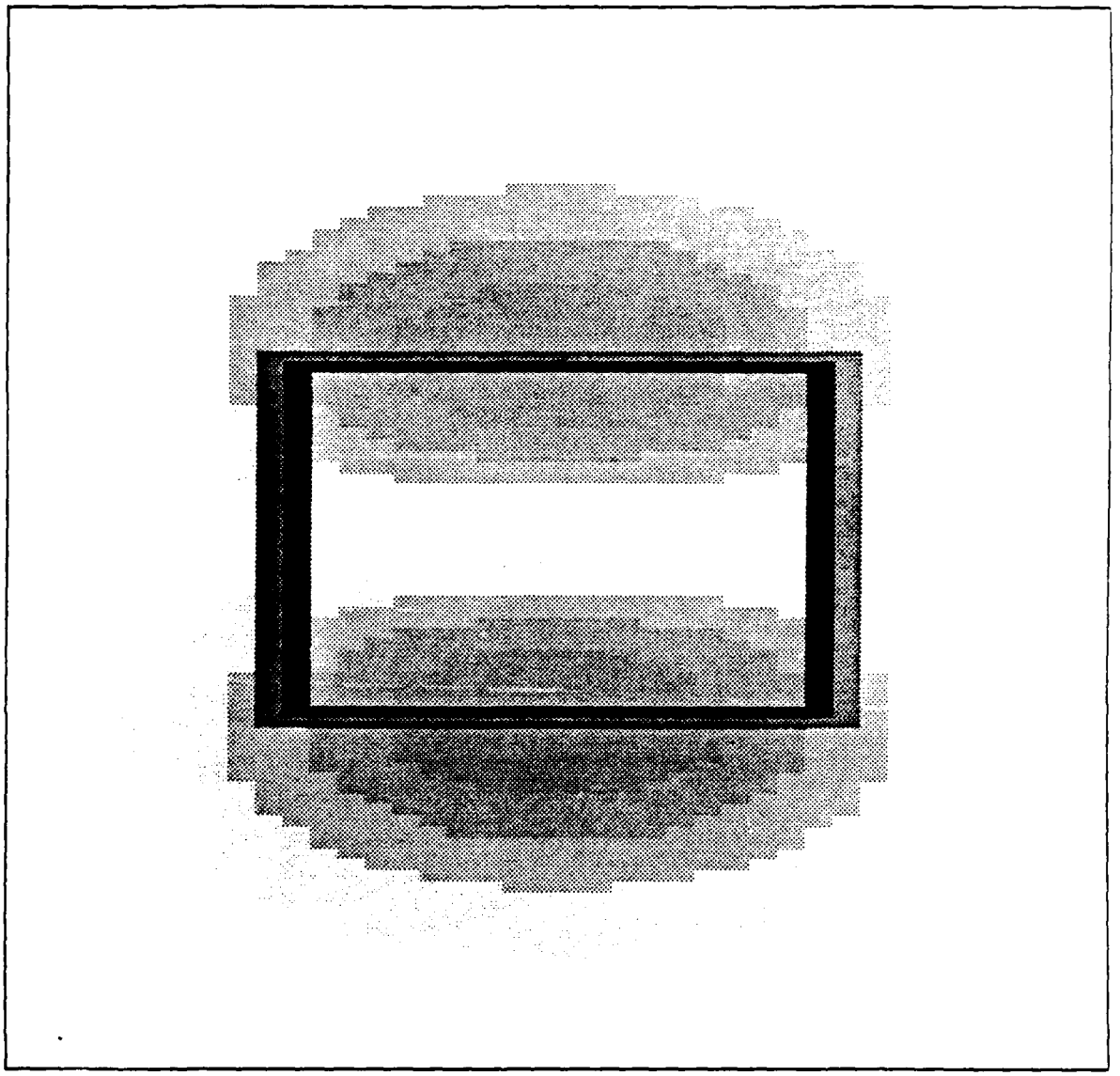
Figure 4e

Figures 4e-4h: grayscale plots of flaws S2, S6, S-4, S-8. The darker rectangle is the actual flaw outline; The lighter rectangle is the flaw as estimated by the program.



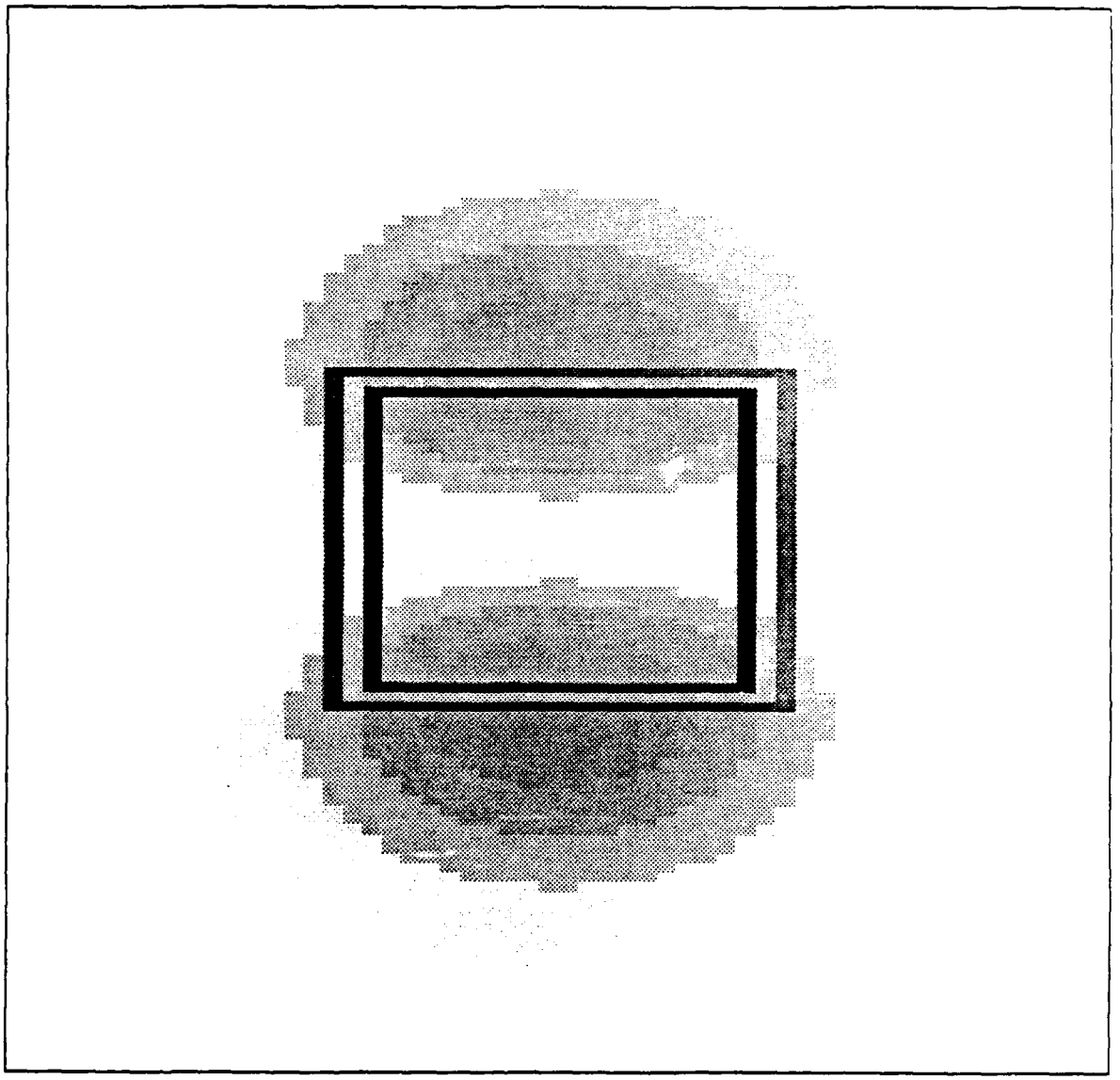
Flaw S6

Figure 4f



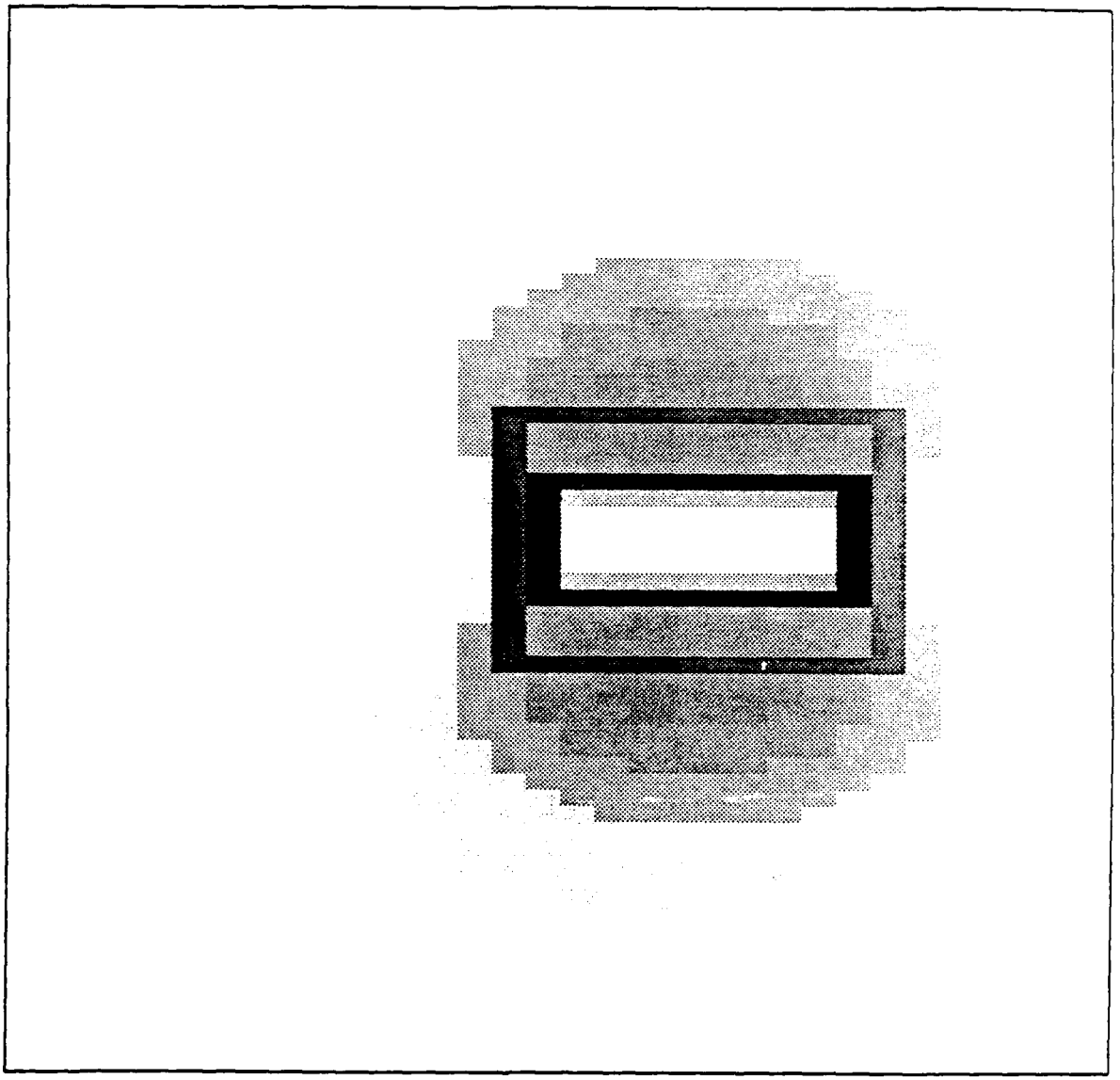
Flaw S-4

Figure 4g



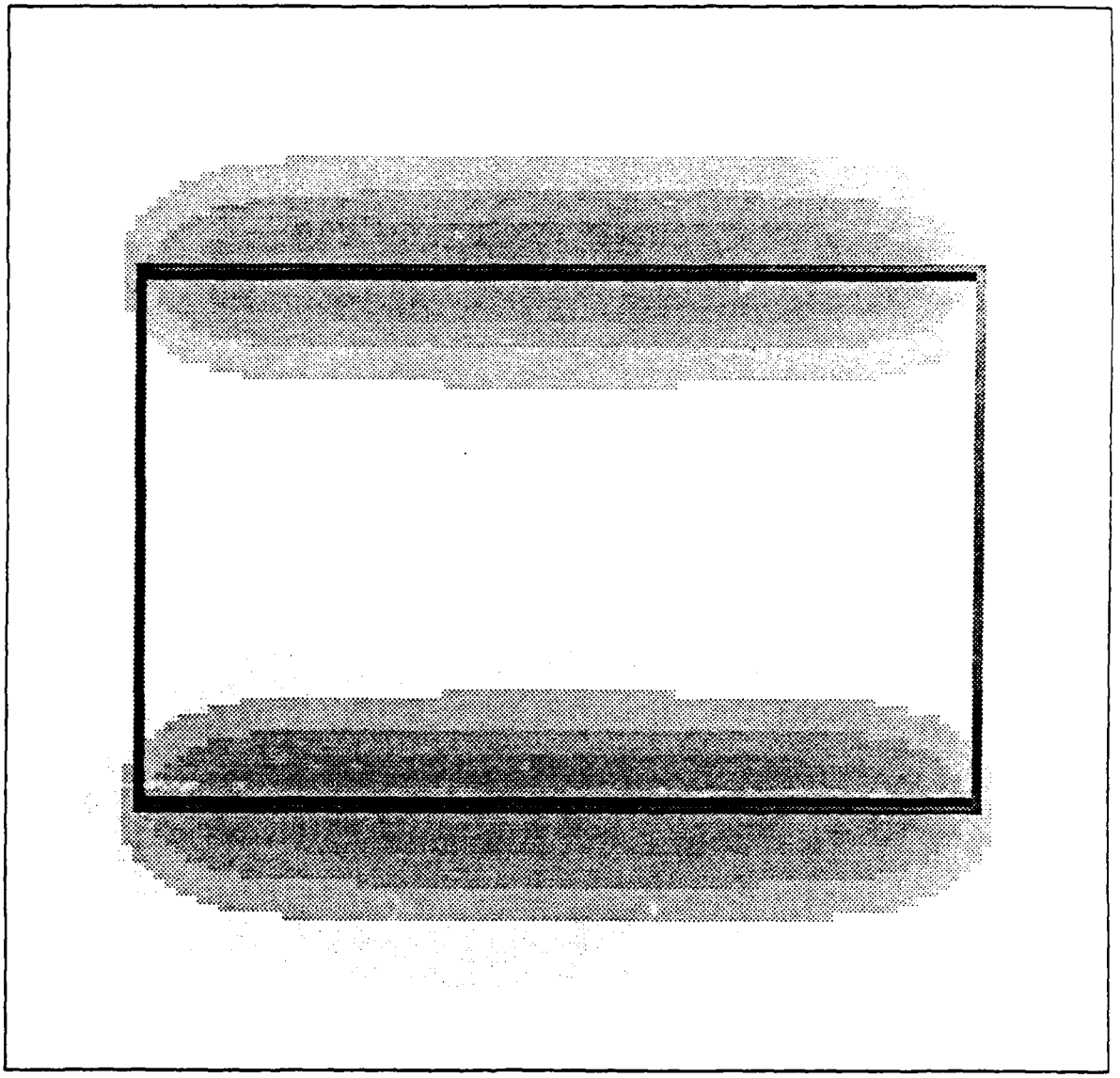
Flow S-8

Figure 4h



Flaw tT6

Figure 3a: grayscale plot of flaw tT6, a small flaw, z = 0.032 in. X phi = 0.0244 in. Dark rectangle is flaw outline, lighter rectangle is estimate. Note overestimate.



Flaw bT6

Figure 3b: grayscale plot of flaw bT6, a large flaw, z = .256 in X phi = 0.195 in. Width of flaw and estimate are the same; the lighter rectangle overwrites the darker on the sides.

```
using flaw h

enter command> r
reading data from file /c/navdat2/cache/hr_fh.500kHz:
    zstart = 0, phistart = 0, skip = 8...
they say data was read in okay.

enter command> t
set to read at zstart = 80, phistart = 160, zlength = 96, philength = 32.

enter command> r
reading data from file /c/navdat2/cache/hr_fh.500kHz:
    zstart = 80, phistart = 160, skip = 1...
they say data was read in okay.

enter command> a
reading data from file /c/navdat2/cache/hr_fh.4MHz, zstart = 80, phistart = 160, skip = 1...
they say data was read in okay.

enter command> gc
enter the corners as they were originally defined> -1 1 -24 25

enter command> p
flaw identified.

enter command> s

enter command> sp
data type: m.
maximum value = 2.629167, [z,phi] = [153,179]
4MHz max = 1.551727, flaw is on the inside.
corner[0] = 2.553593 @ [103,178]
corner[1] = 2.553593 @ [103,180]
corner[2] = 2.553593 @ [152,178]
corner[3] = 2.553593 @ [152,180]
flaw h is within region [102,174], [102,184], [153,174], [153,184]
estimated flaw length = 0.21 inches;  estimated flaw width = 0.07 inches.
actual flaw length = 0.20 inches;  actual flaw width = 0.02 inches.
absolute length error = 0.01 inches;  absolute width error = 0.05 inches.
relative length error = %4.00;  relative width error = %266.67.

enter command> q
```

Figure 6a

Figures 6a-c: programs for Purdue flaw h, c, and e as simulated by the Sabbagh model. These are all long, narrow flaws; note accuracy of length estimate and overestimation of width.

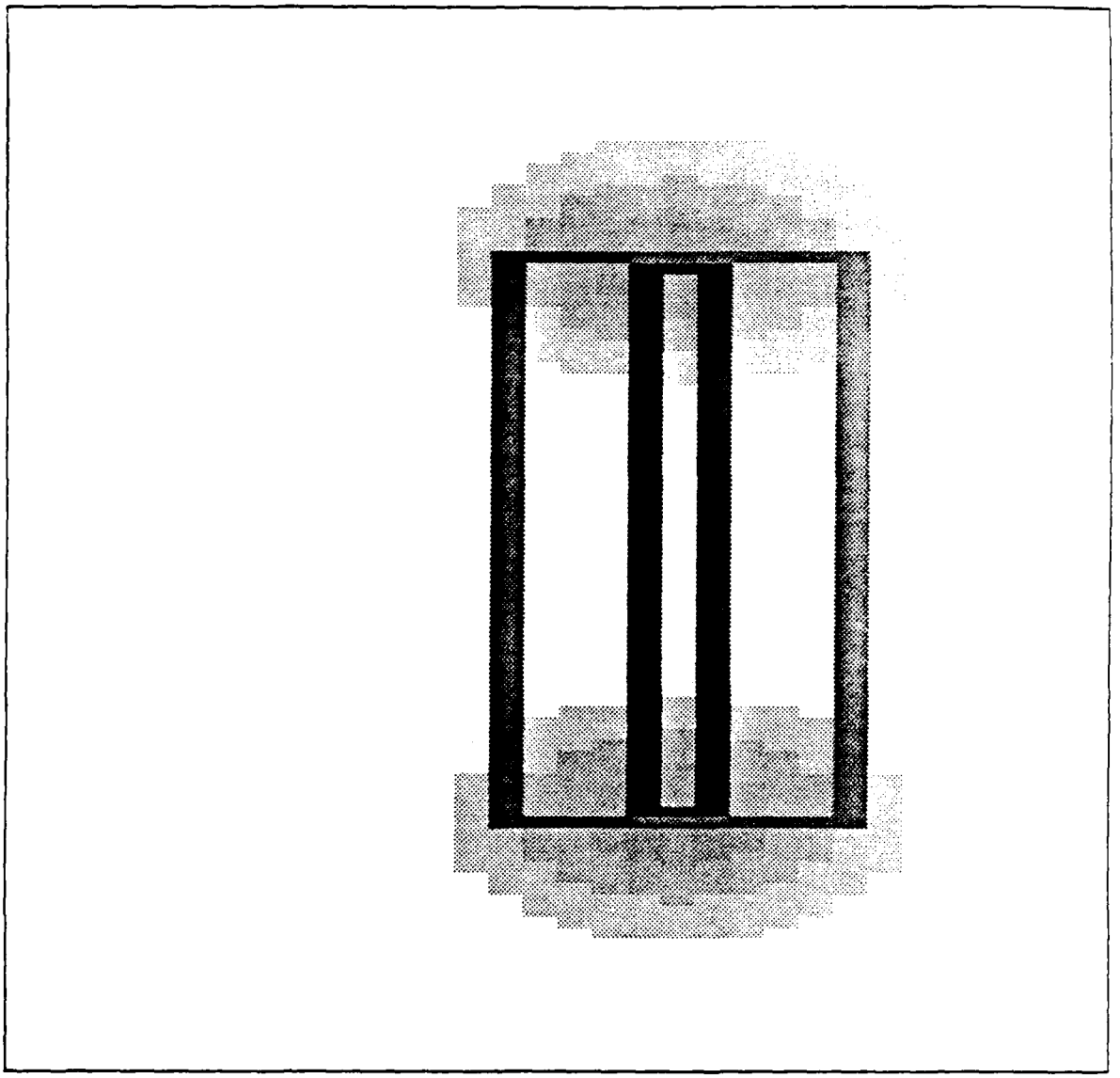
```
using flaw c

enter command> r
reading data from file /c/navdat2/cache/hr_fc.500kHz:
    zstart = 0, phistart = 0, skip = 8...
they say data was read in okay.

enter command> t
set to read at zstart = 72, phistart = 160, zlength = 104, philength = 32.

enter command> r
reading data from file /c/navdat2/cache/hr_fc.500kHz:
    zstart = 72, phistart = 160, skip = 1...
they say data was read in okay.

enter command> a
reading data from file /c/navdat2/cache/hr_fc.4MHz, zstart = 72, phistart = 160, skip = 1...
they say data was read in okay.

enter command> gc
enter the corners as they were originally defined> -1 1 -24 25

enter command> p
flaw identified.

enter command> s

enter command> sp
data type: m.
maximum value = 2.736144, [z,phi] = [153,179]
4MHz max = 1.551520, flaw is on the inside.
corner[0] = 2.602403 @ [103,178]
corner[1] = 2.602403 @ [103,180]
corner[2] = 2.658800 @ [152,178]
corner[3] = 2.658800 @ [152,180]
flaw c is within region [101,174], [101,184], [153,174], [153,184]
estimated flaw length = 0.21 inches;  estimated flaw width = 0.07 inches.
actual flaw length = 0.20 inches;  actual flaw width = 0.02 inches.
absolute length error = 0.01 inches;  absolute width error = 0.05 inches.
relative length error = %6.00;  relative width error = %266.67.

enter command> q
```

Figure 6b

```
using flaw e

enter command> r
reading data from file /c/navdat2/cache/hr_fe.500kHz:
    zstart = 0, phistart = 0, skip = 8...
they say data was read in okay.

enter command> t
set to read at zstart = 80, phistart = 160, zlength = 96, philength = 40.

enter command> r
reading data from file /c/navdat2/cache/hr_fe.500kHz:
    zstart = 80, phistart = 160, skip = 1...
they say data was read in okay.

enter command> a
reading data from file /c/navdat2/cache/hr_fe.4MHz, zstart = 80, phistart = 160, skip = 1...
they say data was read in okay.

enter command> gc
enter the corners as they were originally defined> -1 1 -24 25

enter command> p
flaw identified.

enter command> s

enter command> sp
data type: m.
maximum value = 2.422932, [z,phi] = [148,179]
4MHz max = 1.560291, flaw is on the inside.
corner[0] = 2.157280 @ [103,178]
corner[1] = 2.157280 @ [103,180]
corner[2] = 2.025612 @ [152,178]
corner[3] = 2.025612 @ [152,180]
flaw e is within region [106,174], [106,184], [148,174], [148,184]
estimated flaw length = 0.17 inches;  estimated flaw width = 0.07 inches.
actual flaw length = 0.20 inches;  actual flaw width = 0.02 inches.
absolute length error = -0.03 inches;  absolute width error = 0.05 inches.
relative length error = %-14.00;  relative width error = %266.67.

enter command> q
```

Figure 6c

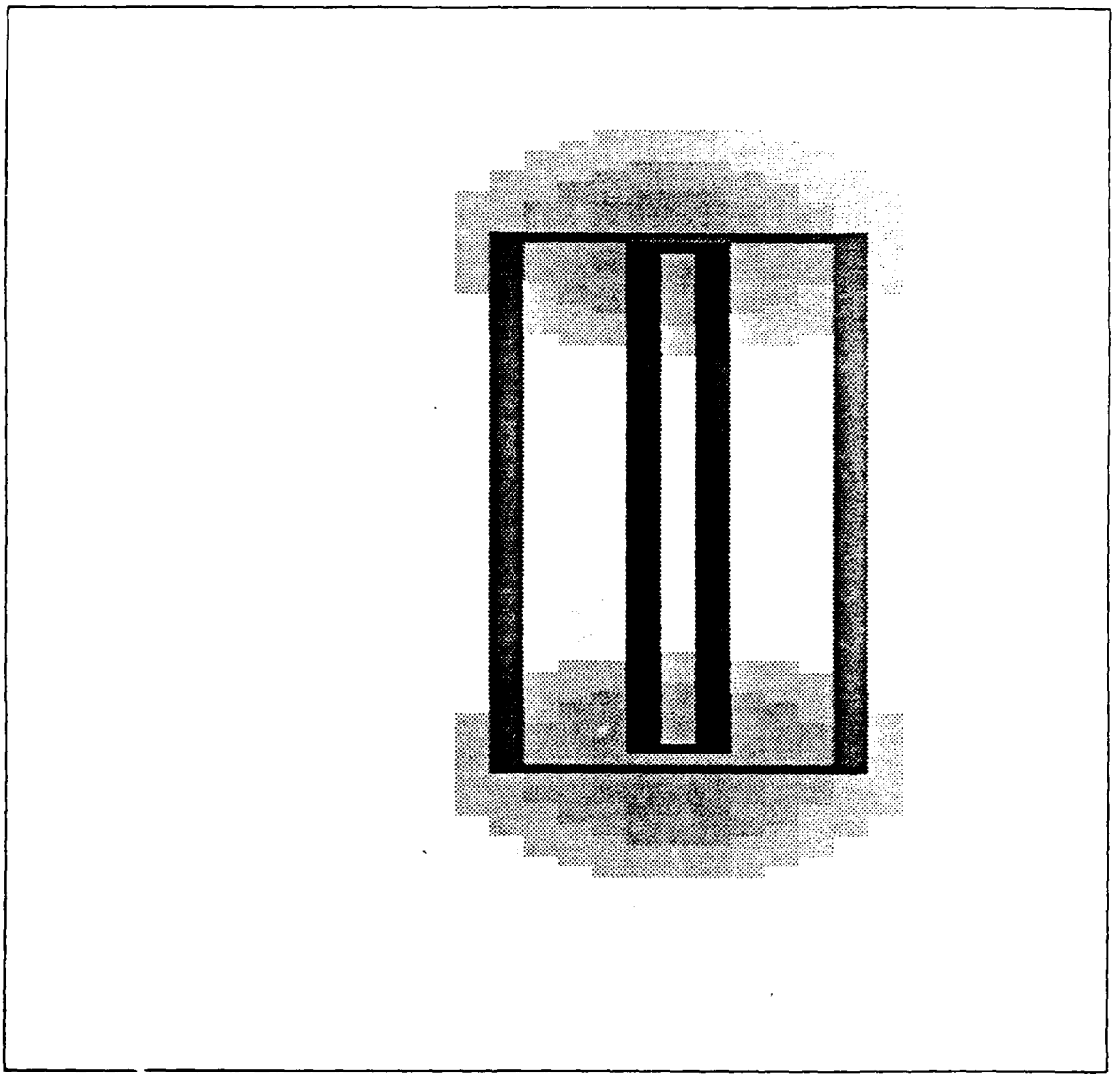
Flaw e is deeper in the middle than on the ends. This causes a length overestimation.



Flaw h

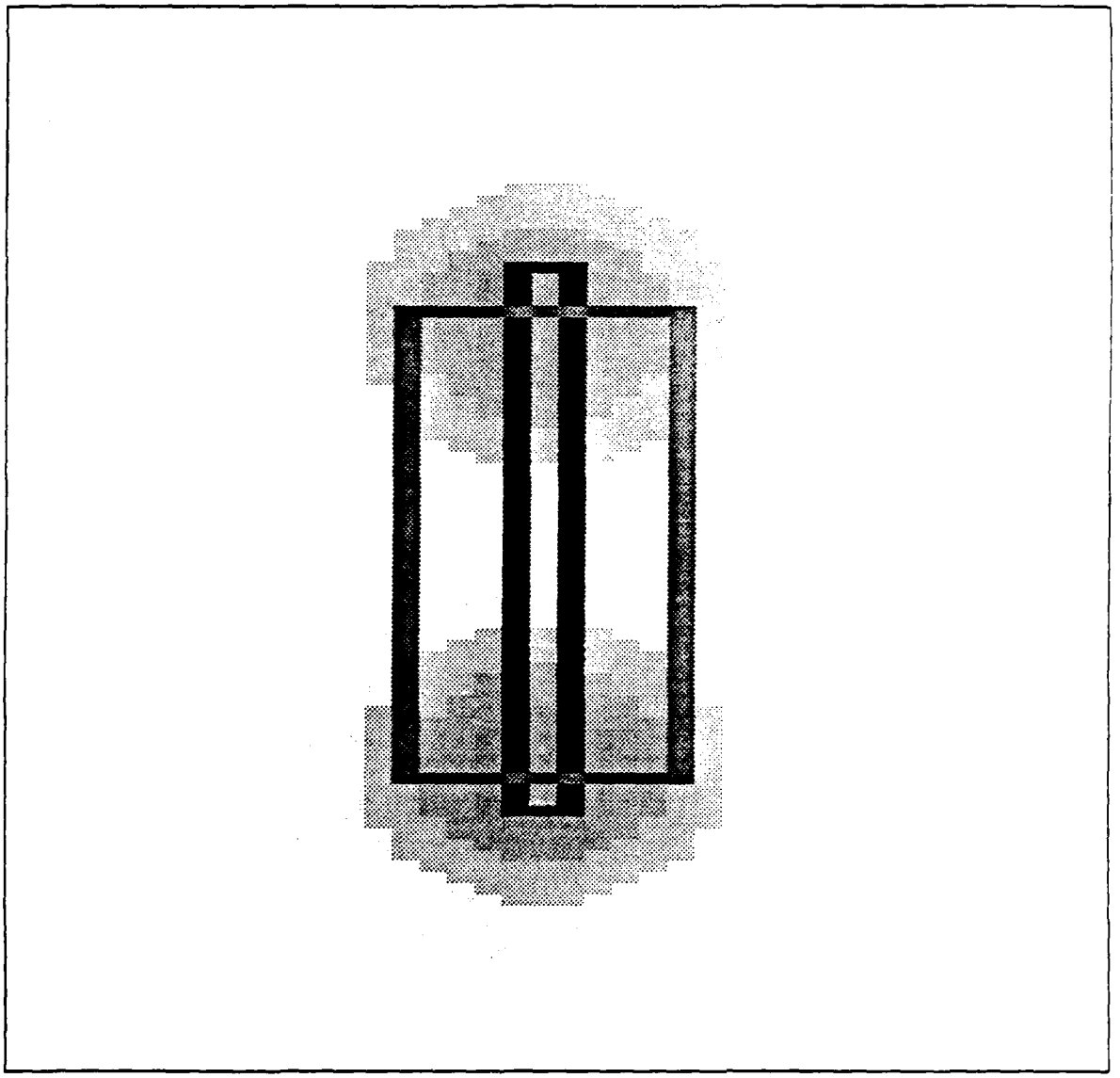
Figure 6d

Figures 6d-f: Grayscale plots of flaws h, c, and e.



Flaw c

Figure 6e



Flow e

Figure 6f

```
Dec  4 09:05 1987  hrad2.list Page 1


using flaw (null)

enter command> n
enter flaw name> /c/navdat2/cache/h_hrad2.500kHz

enter command> r
reading data from file /c/navdat2/cache/h_hrad2.500kHz:
    zstart = 0, phistart = 0, skip = 4...
they say data was read in okay.

enter command> t
set to read at zstart = 28, phistart = 8, zlength = 36, philength = 8.

enter command> r
reading data from file /c/navdat2/cache/h_hrad2.500kHz:
    zstart = 28, phistart = 8, skip = 1...
they say data was read in okay.

enter command> gf
enter flaw width and length> 0.022000 0.200000
enter command> p
flaw identified.

enter command> sp
data type: m.
maximum value = 438.889929, [z,phi] = [38,12]
flaw hrad2 is within region [38,11], [38,13], [55,11], [55,13]
estimated flaw length = 0.18 inches;  estimated flaw width = 0.16 inches.
actual flaw length = 0.20 inches;  actual flaw width = 0.05 inches.
absolute length error = -0.02 inches;  absolute width error = 0.11 inches.
relative length error = %-10.00;  relative width error = %200.00.

enter command> s

enter command> q
```

Figure 7a

Figures 7a-c: Program runs for Purdue data, flaws h, c, and e.

```
using flaw (null)

enter command> n
enter flaw name> /c/navdat2/cache/h_crad.500kHz

enter command> r
reading data from file /c/navdat2/cache/h_crad.500kHz:
    zstart = 0, phistart = 0, skip = 4...
they say data was read in okay.

enter command> t
set to read at zstart = 32, phistart = 8, zlength = 36, philength = 8.

enter command> r
reading data from file /c/navdat2/cache/h_crad.500kHz:
    zstart = 32, phistart = 8, skip = 1...
they say data was read in okay.

enter command> gf
enter flaw width and length> 0.022000 0.200000
enter command> p
flaw identified.

enter command> sp
data type: m.
maximum value = 0.000142, [z,phi] = [40,12]
flaw crad is within region [40,11], [40,13], [59,11], [59,13]
estimated flaw length = 0.20 inches;  estimated flaw width = 0.16 inches.
actual flaw length = 0.20 inches;  actual flaw width = 0.05 inches.
absolute length error = 0.00 inches;  absolute width error = 0.11 inches.
relative length error = %0.00;  relative width error = %200.00.

enter command> s

enter command> q
```

Figure 7b

```
using flaw (null)

enter command> n
enter flaw name> /c/navdat2/cache/h_erad.500kHz

enter command> r
reading data from file /c/navdat2/cache/h_erad.500kHz:
    zstart = 0, phistart = 0, skip = 4...
they say data was read in okay.

enter command> t
set to read at zstart = 24, phistart = 8, zlength = 36, philength = 8.

enter command> r
reading data from file /c/navdat2/cache/h_erad.500kHz:
    zstart = 24, phistart = 8, skip = 1...
they say data was read in okay.

enter command> gf
enter flaw width and length> 0.022000 0.200000
enter command> p
flaw identified.

enter command> sp
data type: m.
maximum value = 0.000094, [z,phi] = [48,12]
flaw erad is within region [36,11], [36,13], [48,11], [48,13]
estimated flaw length = 0.13 inches;  estimated flaw width = 0.16 inches.
actual flaw length = 0.20 inches;  actual flaw width = 0.05 inches.
absolute length error = -0.07 inches;  absolute width error = 0.11 inches.
relative length error = %-35.00;  relative width error = %200.00.

enter command> s

enter command> q
```

Figure 9c

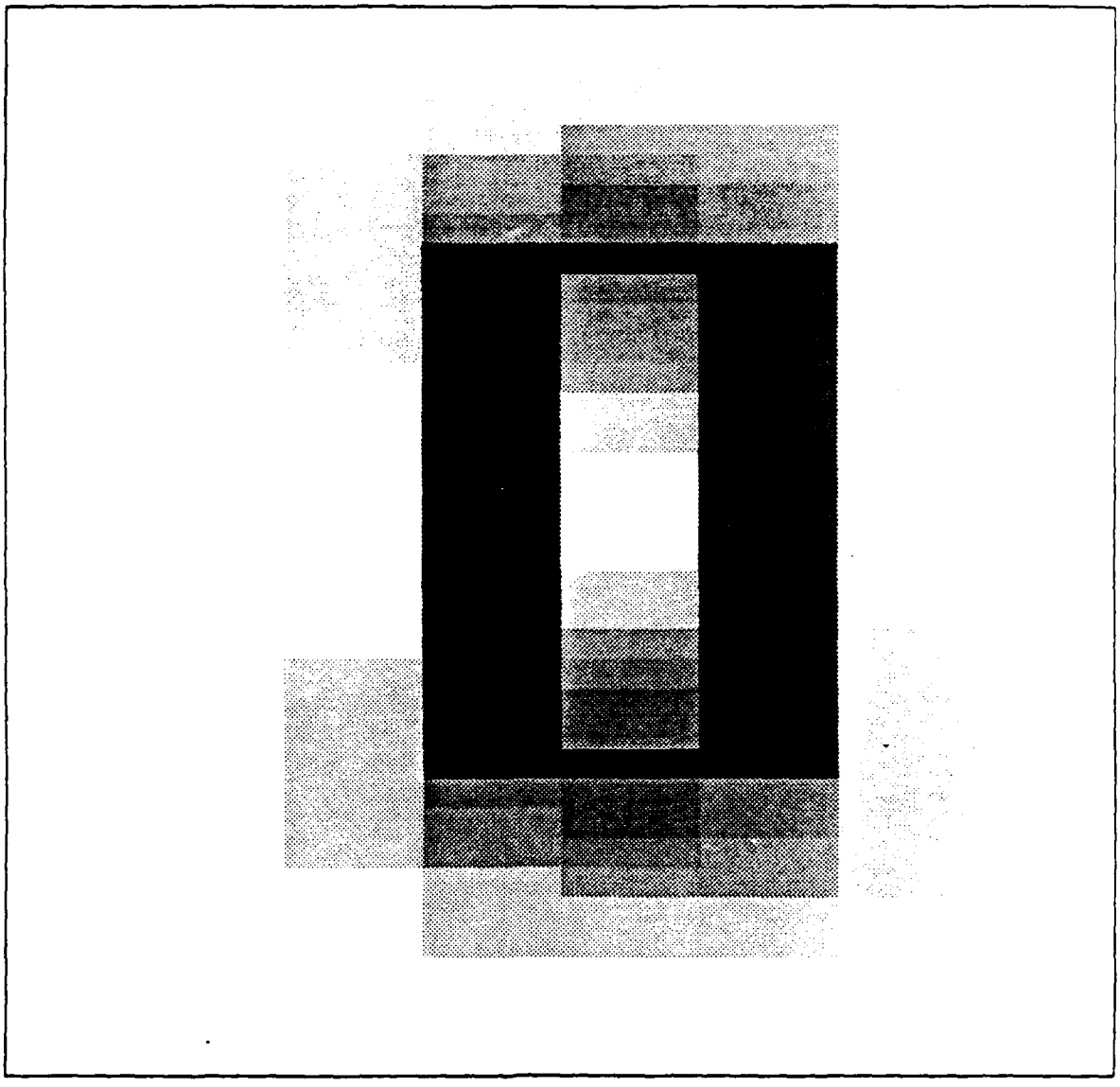
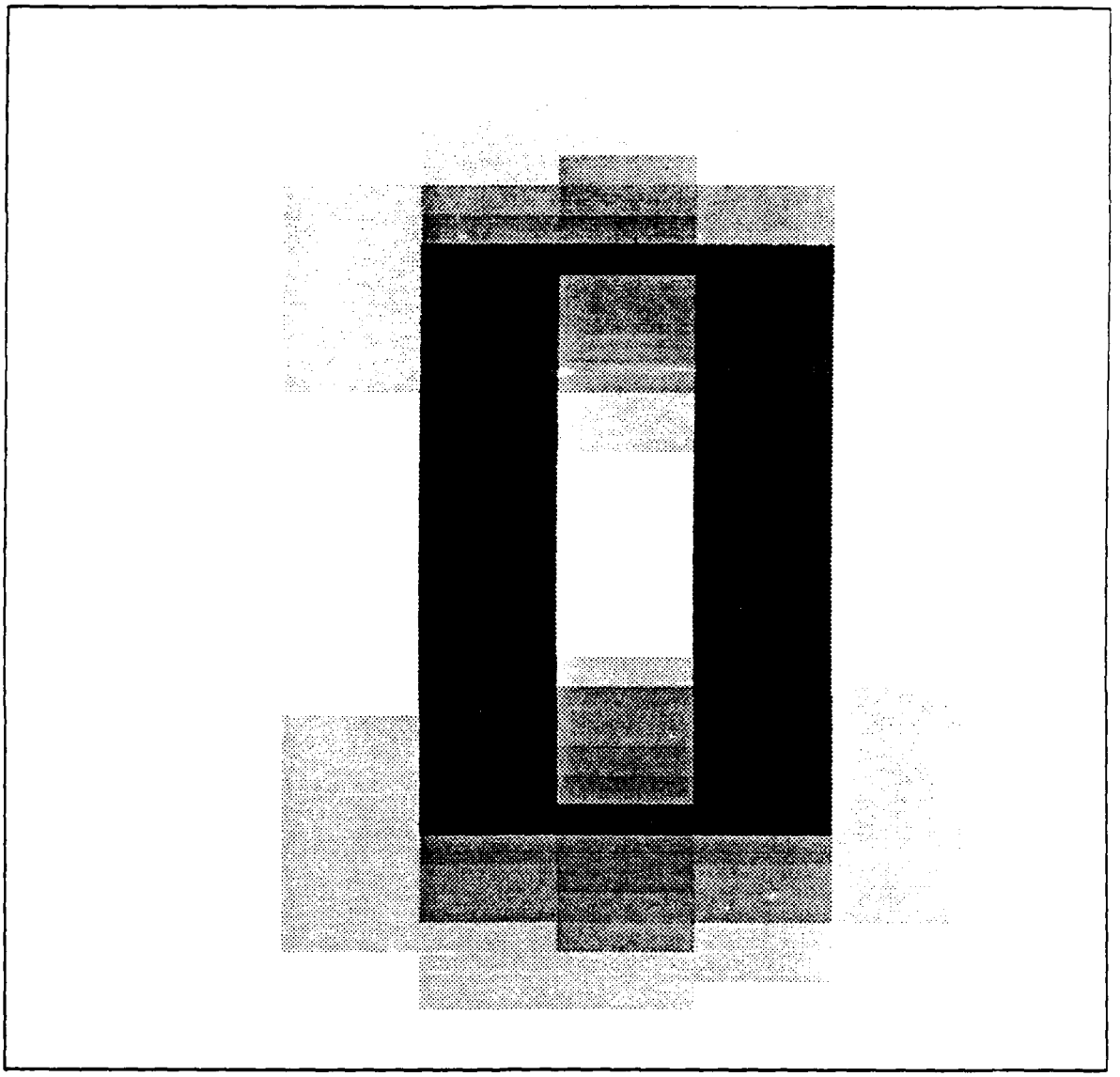


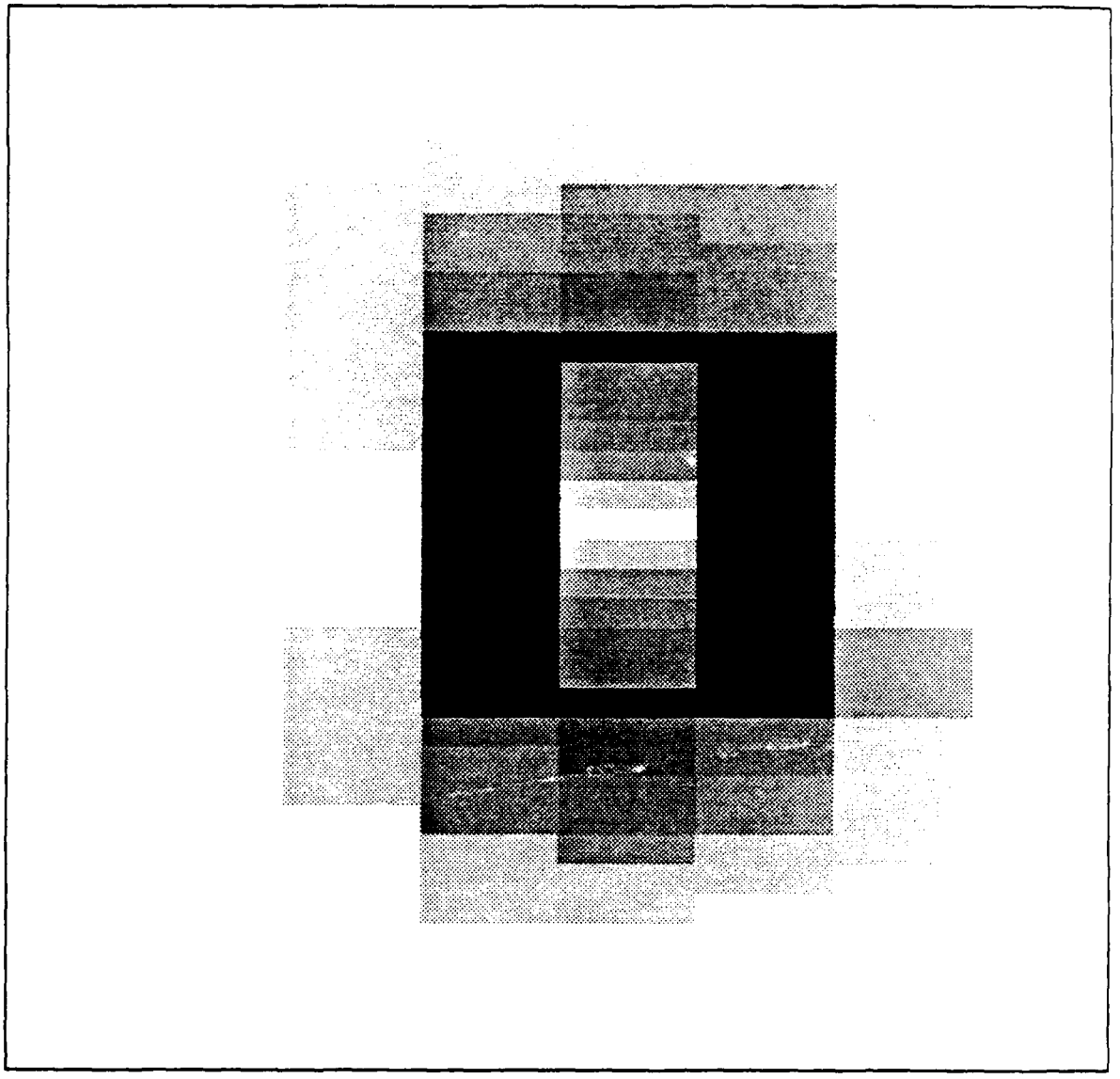
Figure 7d

Figures 7d-f: Grayscale plots of Purdue data, flaws h, c, and e.



Flow crad

Figure 3e



Flaw erad

Figure 7f

```
using flaw x

enter command> r
reading data from file /c/navdat2/cache/hr_fx.500kHz:
    zstart = 0, phistart = 0, skip = 8...
they say data was read in okay.

enter command> t
set to read at zstart = 80, phistart = 0, zlength = 96, philength = 360.

enter command> r
reading data from file /c/navdat2/cache/hr_fx.500kHz:
    zstart = 80, phistart = 0, skip = 1...
they say data was read in okay.

enter command> a
reading data from file /c/navdat2/cache/hr_fx.4MHz, zstart = 80, phistart = 0, skip = 1...
they say data was read in okay.

enter command> gc
enter the corners as they were originally defined> -179 180 -5 6
PHISIZE = 360, ZSIZE = 256
r lengths of sides as zlen philen> -(ZSIZE/2)+1 = -127, (ZSIZE/2) = 128

enter command> p
flaw identified.

enter command> s

enter command> sp
data type: m.
maximum value = 0.046969, [z,phi] = [141,0]
4MHz max = 0.046969, flaw is on the outside.
flaw is between 122 and 133 along z axis.
flaw is of uniform thinning type, found between 114 and 141.
estimated flaw length = 0.11 inches.
actual flaw length = 0.05 inches.
absolute length error = 0.06 inches.
relative length error = %133.33.

enter command> q
```

Figure 9a

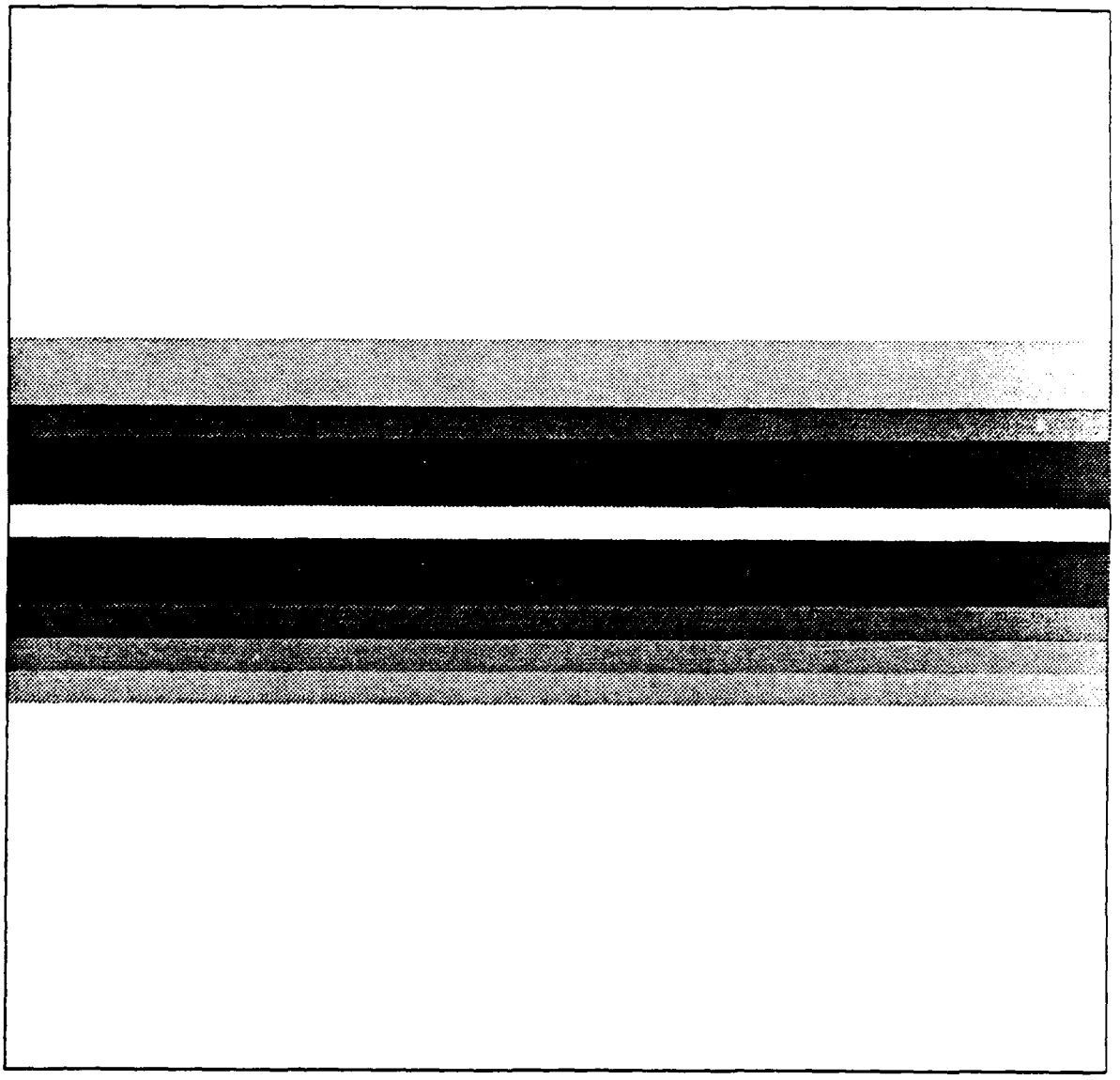
Figures 9a-c: Program runs for three uniform thinning flaws. Flaw x (0.0576 in. deep around the outside), flaw x1 (0.096 in. deep around the outside), flaw x2 (0.096 in. deep around the inside).

```
using flaw x1

enter command> r
reading data from file /c/navdat2/cache/hr_fx1.500kHz:
    zstart = 0, phistart = 0, skip = 8...
they say data was read in okay.

enter command> t
set to read at zstart = 64, phistart = 0, zlength = 128, philength = 360.

enter command> r
reading data from file /c/navdat2/cache/hr_fx1.500kHz:
    zstart = 64, phistart = 0, skip = 1...
they say data was read in okay.

enter command> a
reading data from file /c/navdat2/cache/hr_fx1.4MHz, zstart = 64, phistart = 0, skip = 1...
they say data was read in okay.

enter command> gc
enter the corners as they were originally defined> -179 180 -19 20
PHISIZE = 360, ZSIZE = 256
r lengths of sides as zlen philen> -(ZSIZE/2)+1 = -127, (ZSIZE/2) = 128

enter command> p
flaw identified.

enter command> s

enter command> sp
data type: m.
maximum value = 0.093718, [z,phi] = [107,0]
4MHz max = 0.093718, flaw is on the outside.
flaw is between 108 and 147 along z axis.
flaw is of uniform thinning type, found between 107 and 148.
estimated flaw length = 0.17 inches.
actual flaw length = 0.16 inches.
absolute length error = 0.01 inches.
relative length error = %5.00.

enter command> q
```

Figure 8b

```
using flaw x2

enter command> r
reading data from file /c/navdat2/cache/hr_fx2.500kHz:
    zstart = 0, phistart = 0, skip = 8...
they say data was read in okay.

enter command> t
set to read at zstart = 72, phistart = 0, zlength = 104, philength = 360.

enter command> r
reading data from file /c/navdat2/cache/hr_fx2.500kHz:
    zstart = 72, phistart = 0, skip = 1...
they say data was read in okay.

enter command> a
reading data from file /c/navdat2/cache/hr_fx2.4MHz, zstart = 72, phistart = 0, skip = 1...
they say data was read in okay.

enter command> gc
enter the corners as they were originally defined> -179 180 -19 20
PHISIZE = 360, ZSIZE = 256
r lengths of sides as zlen philen> -(ZSIZE/2)+1 = -127, (ZSIZE/2) = 128

enter command> p
flaw identified.

enter command> s

enter command> sp
data type: m.
maximum value = 10.547817, [z,phi] = [107,0]
4MHz max = 10.547817, flaw is on the inside.
flaw is between 108 and 147 along z axis.
flaw is of uniform thinning type, found between 107 and 148.
estimated flaw length = 0.17 inches.
actual flaw length = 0.16 inches.
absolute length error = 0.01 inches.
relative length error = %5.00.

enter command> q
```

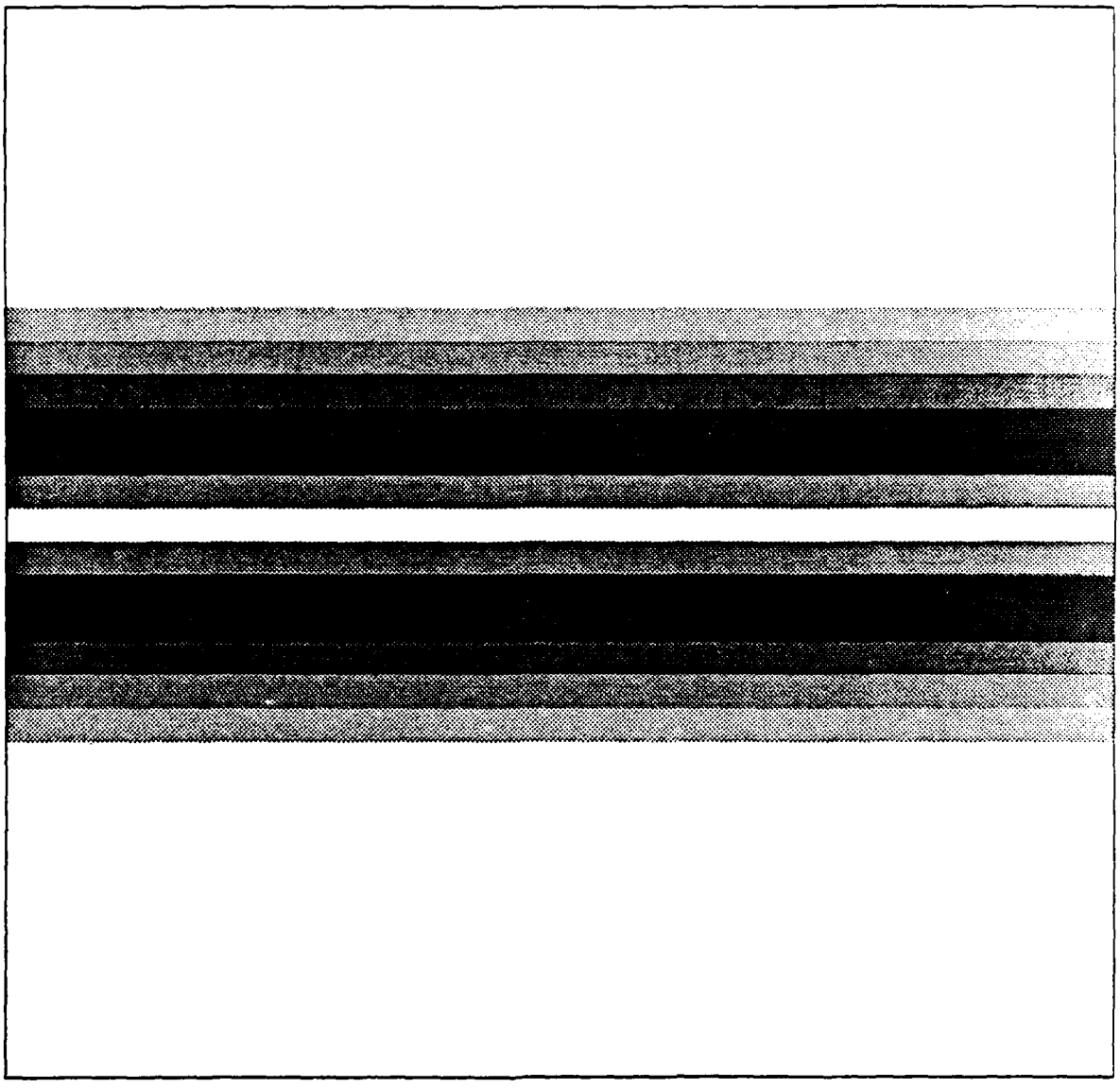
Figure 8c



Flaw x

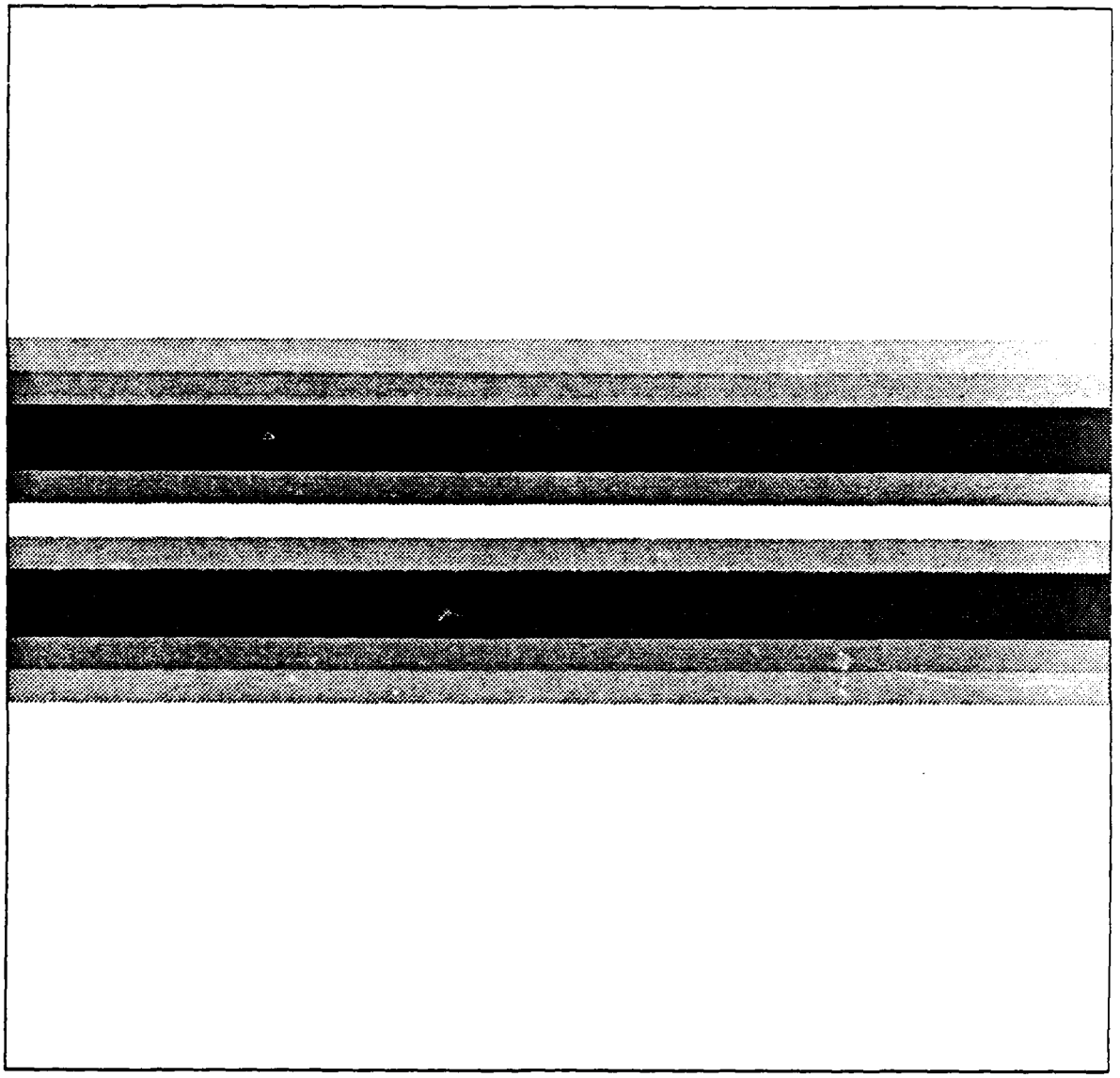
Figure 6d

Figure 6d-f: grayscale plots of flaws x, x1, x2.



Flaw x1

Figure 6a



Flow x2

Figure 8f

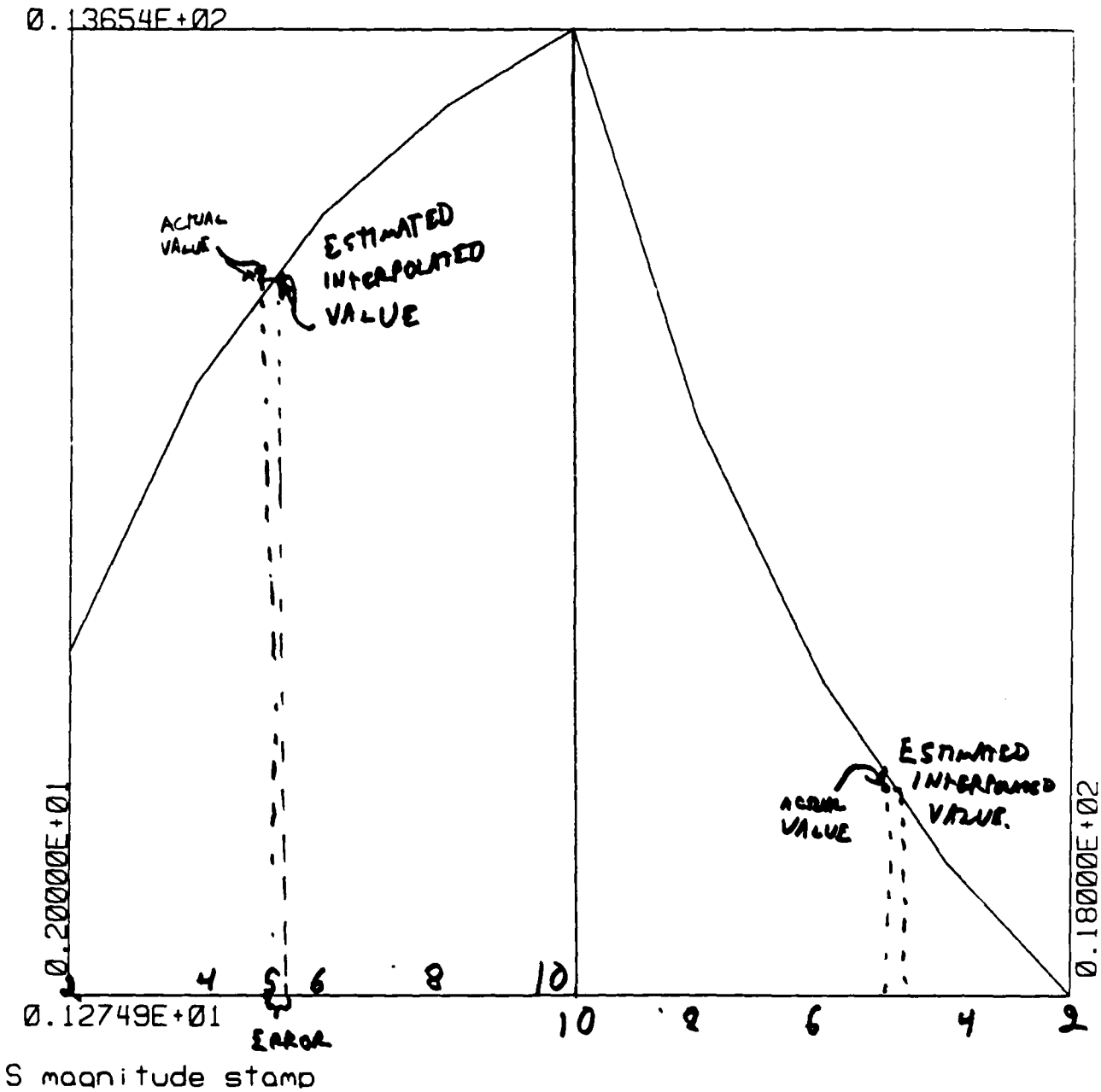


Figure 1: the effect of interpolation error. Flaws on the inside are estimated to be deeper than they are; flaws on the outside are estimated to be shallower than they are. This diagram shows error due to depth interpolation only; the effect is multiplied when the width and length also lie between table values.

```
<enter flaw width> 30
enter flaw length> 32
enter value at 40kHz> 13.3
enter value at 4MHz> 5.10
flaw is 6.583090 deep on the inside.

(flaw is 6.0 deep on inside.)


enter flaw width> 20
enter flaw length> 32
enter value at 40kHz> 10.3
enter value at 4MHz> 4.85
flaw is 5.101852 deep on the inside.

(flaw is 5.0 deep on the inside.)


enter flaw width> 20
enter flaw length> 26
enter value at 40kHz> 10.1
enter value at 4MHz> 4.74
flaw is 6.530351 deep on the inside.

(flaw is 6.0 deep on the inside.)


enter flaw width> 22
enter flaw length> 32
enter value at 40kHz> 5.66
enter value at 4MHz> 0.00338
flaw is 4.071072 deep on the outside.


(flaw is 6.0 deep on the outside.)


enter flaw width> 20
enter flaw length> 32
enter value at 40kHz> 4.06
enter value at 4MHz> 0.000539
flaw is 2.918455 deep on the outside.

(flaw is 5.0 deep on the outside.)


enter flaw width> 20
enter flaw length> 32
enter value at 40kHz> 4.06
enter value at 4MHz> 0.0000539
flaw is 2.918455 deep on the outside.

(flaw is 5.0 deep on the outside.)


enter flaw width> 20
enter flaw length> 48
enter value at 40kHz> 6.53
enter value at 4MHz> 0.0034
flaw is 4.165312 deep on the outside.

(flaw is 6.0 deep on the outside.)


enter flaw width> 30
enter flaw length> 48
enter value at 40kHz> 0.00352
enter value at 4MHz> 4
flaw is 0.000903 deep on the inside.

(flaw is 6.00 deep on the outside.)


enter flaw width> 30
enter flaw length> 32
enter value at 40kHz> 6.71
enter value at 4MHz> 0.000352
flaw is 4.155879 deep on the outside.

(flaw is 6.0 deep on the outside.)
```

Figure 10: Results of program runs for various flaws showing estimated depths; correct depths are included.

```
enter flaw width> 30
enter flaw length> 48
enter value at 40kHz> 8.35
enter value at 4MHz> 0.00363
flaw is 4.346963 deep on the outside.

(flaw is 6.0 deep on the outside.)


enter flaw width> 20
enter flaw length> 48
enter value at 40kHz> 5.03
enter value at 4MHz> 0.000553
flaw is 3.106542 deep on the outside.

(flaw is 5.0 deep on the outside.)


enter flaw width> 22
enter flaw length> 32
enter value at 40kHz> 11.8
enter value at 4MHz> 4.93
flaw is 6.304555 deep on the inside.

(flaw is 6.0 deep on inside.)


enter flaw width> 20
enter flaw length> 32
enter value at 40kHz> 10.3
enter value at 4MHz> 4.85
flaw is 5.101852 deep on the inside.


(flaw is 5.0 deep on the inside.)


enter flaw width> 20
enter flaw length> 48
enter value at 40kHz> 13.0
enter value at 4MHz> 4.97
flaw is 6.630630 deep on the inside.


(flaw is 6.0 deep in the inside.)


enter flaw width> 30
enter flaw length> 32
enter value at 40kHz> 13.3
enter value at 4MHz> 5.10
flaw is 6.583090 deep on the inside.

(flaw is 6.0 deep on the inside.)


enter flaw width> 30
enter flaw length> 48
enter value at 40kHz> 15.6
enter value at 4MHz> 5.25
flaw is 7.195123 deep on the inside.

(flaw is 6.0 deep on the inside.)


enter flaw width> 20
enter flaw length> 48
enter value at 40kHz> 11.8
enter value at 4MHz> 4.97
flaw is 5.453442 deep on the inside.

(flaw is 5.0 deep on the inside.)
```

Figure 10 (cont.)

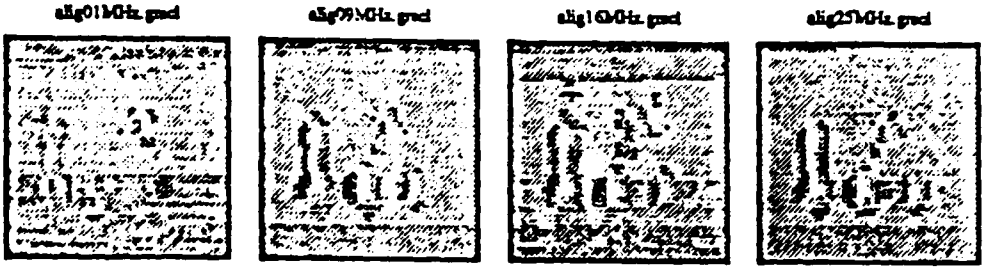


Figure 11: a field showing real data.

Figure 12: a field showing magnitude data.

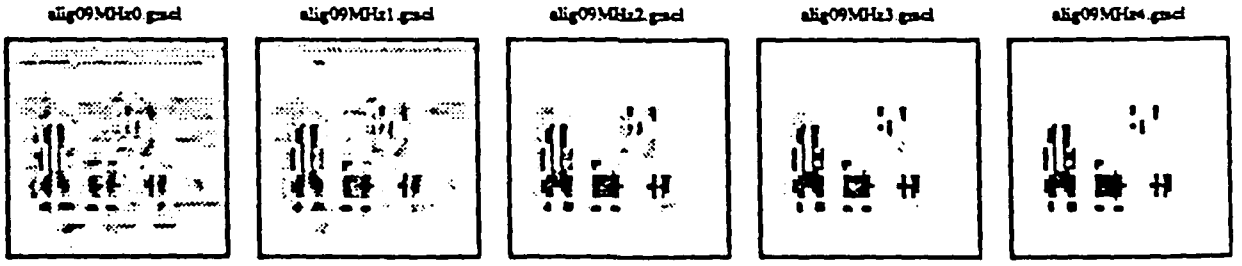


Figure 13: increasing the resolution to make flaws stand out.